The Report Committee for Ricardo Ramirez
certifies that this is the approved version of the following report:

Semi-formal Verification of Analog Mixed Signal Systems Using Multi-domain Modeling Languages

APPROVED BY

SUPERVISING COMMITTEE:

Jacob Abraham, Supervisor

Adnan Aziz

Semi-formal Verification of Analog Mixed Signal Systems Using Multi-domain Modeling Languages

by

Ricardo Ramirez, B.S., M.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2013

Dedicated to the Latin American Indigenous and Afro-descendants peoples whose rights and freedoms have greatly been usurped through the centuries.

Acknowledgments

I want to thank all my professors , whose patience, achievements and support have always inspired me to keep learning. I was specially motivated by the cutting edge research work and practical teaching methodologies carried out by Professor Jacob Abraham, Professor Adnan Aziz, Professor Ranjit Gharpurey, Mark McDermott, Professor Andreas Gerstlauer, and Professor Arjang Hassibi.

I am thankful for the chance I was given to be part of the Master of Science program at the University of Texas at Austin. I have had some of the brightness minds and active professionals in the field as my professors, peers and tutors. Many thanks go as well to all the UT Center for Lifelong Engineering Education staff for their endless support.

I as well want to thank all my classmates whose experiences and insights about the different topics we discussed over the length of the program made me aware of how rich their knowledge and skills were.

Finally, I want to thank my family and friends whose support has always been close to my heart and has made me climb the steepest hills of life.

Semi-formal Verification of Analog Mixed Signal Systems Using Multi-domain Modeling Languages

Ricardo Ramirez, M.S.E.
The University of Texas at Austin, 2013

Supervisor: Jacob Abraham

The verification of analog designs has been a challenging task for a few years now. Several approaches have been taken to tackle the main problem related to the complexity that such task presents to design and verification teams. The methodology presented in this document is based on the experiences and research work carried out by the Concordia University's Hardware Verification and the U. of Texas' IC systems design groups.

The representation of complex systems where different interactions either mechanical or electrical take place requires an intricate set of mathematical descriptions which greatly vary according to the system under test. As a simple and very relevant example one can look at the integration of RF-MEMS as active elements in System-On-Chip architectures. In order to tackle such heterogeneous interaction for a consistent model, the use of stochastic hybrid models is described and implemented for very simple examples using high level modeling tools for a succinct and precise description.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Formal analog verification has been a very elusive technical goal for many years now. Several methodologies have been proposed with a wide set of results. Thanks to the work of the Hardware Verification Group at Concordia University a very interesting approach to achieve such a technical grail has partially been implemented.[62]

The methodology for a proper analog design verification requires more than one path for its implementation. Several research works have shown that there is not an unique approach for a complete solution. The combination of several analysis approaches and their respective enablers (software and modeling tools) can bring a coherent solution to the analog verification challenge.

Most of the elements for mathematical modeling of electric circuit behaviors tend to gravitate around the Kirchhoff Laws given their simplicity. Those laws were derived from the formal Maxwell descriptions and are assumed to work only in lumped elements.[11]

A comprehensive technique for modeling analog systems will require a correct description of all the available cases for the elements present in the system. The integration of complex where system-on-chip architectures make

use of elements from the Micro Electro-Mechanical Systems (MEMS) realm requires a more detailed and realistic modeling process. RF front-ends have MEMS associated to fundamental blocks such as Antennas, RF filters, and phase shifters. Figure 1.1 shows an example of a simple architecture used to design a reconfigurable RF Front-End.

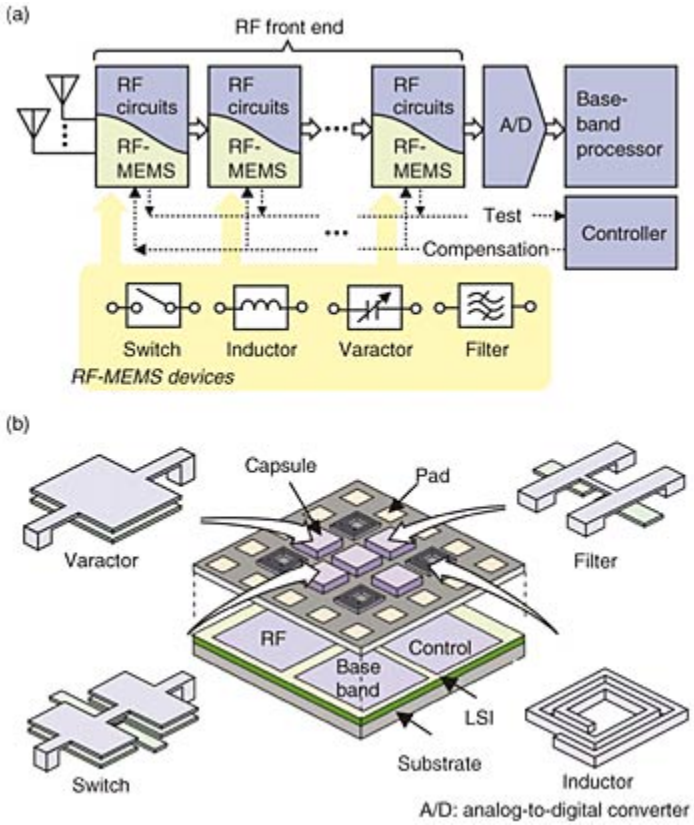

Figure 1.1: MEMS for reconfigurable RF front end:(a) Block diagram. (b) Physical implementation[34]

At the moment, available verification methodologies do not provide a suitable platform for modeling and simulation of the interaction between the

main circuitry body and the above mentioned MEMS elements. Simulations are usually carried out as a fragmented set of multiple descriptions which are linked with abstract connections.

Having a common verification framework will guarantee the development of top-down architectures where the final test-bench is completely independent from the physical setup and a richer stimulus environment can be applied.

The work presented in this report is a mere introduction to the capabilities of this type of verification approach. Circuit simulation has been carried out using Modelica[1] and QUCS[2] environments as a framework [5] [14]

1.1 Proposed Approach

The use of object-oriented programming and abstract definitions can greatly improve the way in which analog systems are analyzed and verified. Elements related to the model of simple behavioral abstractions can easily be linked to software object-oriented models.[14] Paper-pen proof assured functionality in systems with low to moderate complexity. However, integration of new technologies and processing paradigms made the complexity of such analysis grow in a dramatic manner. Borrowing from software verification methodologies a succinct but effective plan for analog verification can be laid out. Most of the structures with heterogeneous components can be labeled

[1] Open Modelica
[2] Quite Universal Circuit Simulator

as Hybrid systems. Analysis of such systems can be done using descriptions based on hybrid theorem provers which can guarantee a seamless transition from algebraic descriptions towards a more likely scenario where relational equations model behavioral answers for the system.[50] Among some of the tools available for verification, KeYmaera is one of the most relevant ones considering its applicability in Hybrid Systems.[51] Modeling tools such as Open-Modelica provide the required framework to describe hybrid models. It uses conditional expressions for conditional models and event-driven equations for discontinuities.[19]

Figure 1.2 presents a block diagram description for the Stochastic Hybrid Systems. Stochastic hybrid systems are useful to describe processes where behaviors can be represented using heterogeneous modeling descriptions. For example, *Continous dynamics* for differential equations, *Discrete dynamics* for control decisions, and *Stochastic dynamics* for uncertainty and probabilistic behaviors.

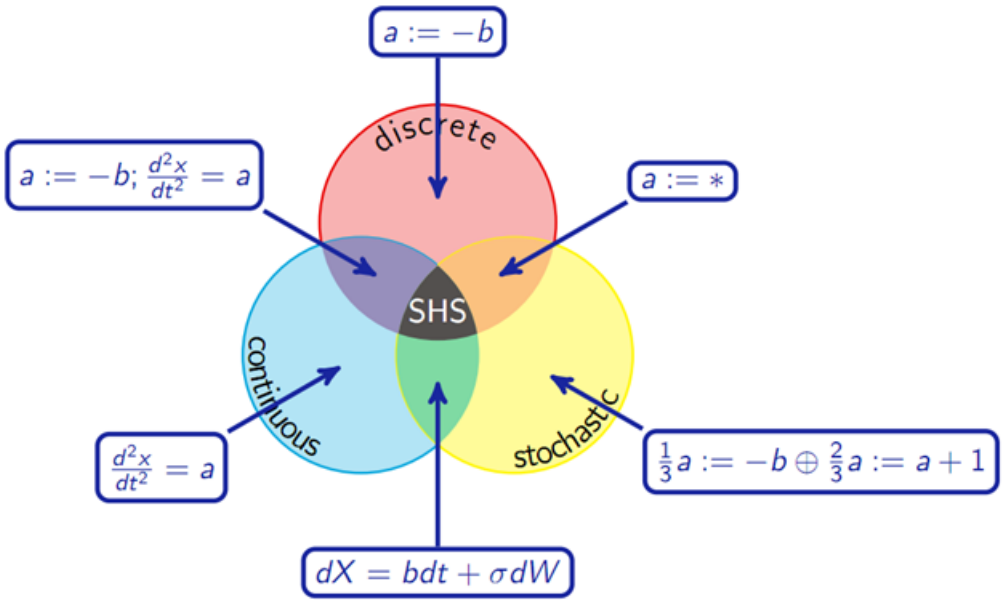

Figure 1.2: Model of Stochastic Hybrid Systems

Chapter 2

Fundamental Concepts

2.1 Process Algebras

Process algebras can be defined as a method for studying concurrent processes. Usually the tools implemented for such study are algebraic languages where processes can be specified with their respective formulation of statements, while an accurate description of calculi for their verification of the same statements is provided. [21]

Process algebras are formal languages with formal syntax. They have semantics for specifying and reasoning about different systems. They allow describing the behavior of processes using singular models to provide a framework for higher complexity systems composition. Process algebras can be used as theoretical frameworks for formal specification and analysis of behavior of various systems, and as such, are ideal methods for the verification of analog integrated circuits. [26]

Algebra of Communicating Processes (ACP) is a hybrid process where algebra with propositional signals and continuous relative timing are interlaced. It is important to highlight that this arrangement provides additional equational axioms that allow deriving equations based on real analysis.

Using the *Structured Operational Semantics* (SOS) style many well-defined process algebras have been developed. This type of modeling methodology has been used in system level design languages for verification of correctness in component integration processes (i.e. their interactions).[56] A simple example is shown in Figure 2.1.

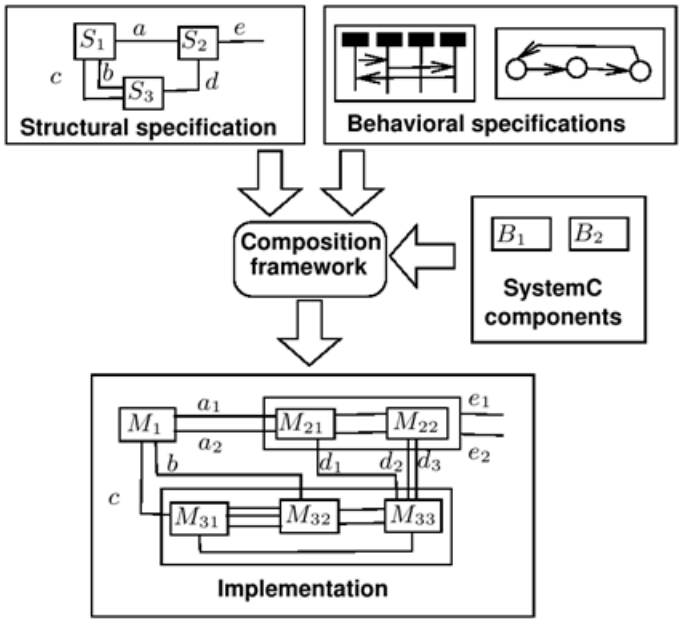

Figure 2.1: Process Algebra - Component Composition

Structural specification provides the framework for the description of components, channels, events, shared variables and their respective connections. The *Behavioral specification* provides the scenarios where the sequence of events can be observed. In Figure 2.1, the *implementation of the components* is carried out using SystemC. For this work, Object-Modelica will be used as the modeling setup. *Verification of logical correctness* comes as a modular

arrangement of abstractions.

2.2 Hybrid Automata

Hybrid automata are formal models that describe dynamic systems including both discrete and continuous behaviors.[46][59] A simple example is the temperature control for an automated system where one wants to vary the temperature according to a set of determined values. The problem can then be described as a set of finite states with transition variables and attached flow conditions. Figure 2.2 presents a state flow diagram for the temperature control model.

Hybrid automata are some of the underlying structures present in *Algebra of Communicating Processes* (ACP) which allows back and forth transitions for better analysis. This characteristic allows verification of ACP-based specifications using hybrid automaton-based tools such as PHAVer and HyTech. The Structured Operational Semantics (SOS) style used by ACP allows relating a hybrid transition to a process that can be defined as a state.

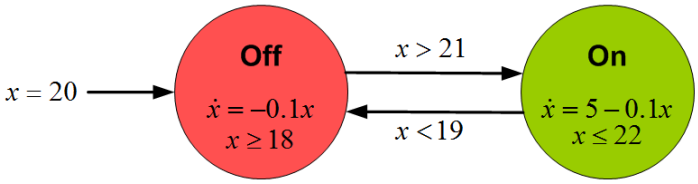

Figure 2.2: Temperature control using hybrid automata[27]

In the temperature example, there are two well defined states. They depend on a set temperature. In this example, 20°C is set as the goal tem-

State	Rate of change	Condition
OFF	$\dot{x} = -0.1x$	$x \geq 18$
ON	$\dot{x} = 5 - 0.1x$	$x \leq 22$

Table 2.1: Hybrid automata - Temperature Example

Init State	Condition	Next State
OFF	$x < 19$	ON
ON	$x > 21$	OFF

Table 2.2: Hybrid automata - Temperature Example - State flow

perature. If the temperature goes beyond 21°C the controller is turned OFF. If the temperature goes below 19°C the controller bounces back to ON state. Tables 2.1 and 2.2 presents the required equations to describe the states flow.

2.2.1 Formal Definition

A hybrid automaton consists of several parts: variables, control graph, flow conditions (initial state, invariant setup, and jump) and finite events.

- Variables: Provide information about finite states, continuous changes and resulting states after discrete changes.
- Control graph: Provide a visual and an intuitive tool to describe directed actions. It provides information relevant to control modes and control switches (for discrete states and discrete dynamics respectively).
- Flow conditions: provide information about initial setup, invariant clauses, and control flow.

- Discrete Changes: jump conditions and finite events are included as part of the automatas description.

2.2.1.1 State Space of Hybrid Automata

State space for hybrid automata cannot be enumerated, i.e., is *uncountable*. It can, however, be studied using representations in the finite symbolic realm. Variables can be defined as part of a specific domain (e.g. real numbers) with finite symbolic representation of the possibilities implied in the state space. For instance, x variable can be set in the real number domain. It can then have infinite points moving from 1 to 1000 represented as $1 \leq x \leq 1000$.

2.2.1.2 Labeled Transition Systems

Labeled transitions allow describing transition relations among the different possible states for a given system. The relationships between states can be described as a set of transitions which are represented in a specific domain or region. Transitions can as well be described as a time-dependent abstraction. Timed transitions allow passing information related to source, target and flow duration to the system. Time-Abstract transitions allow capturing information about the duration of the flow.

2.2.1.3 Live Semantics and Transition Systems

Hybrid automata are usually set to cover conditions where infinite sequences are considered. Transitions under such premises do not converge in

time and as such there is a perennial time divergence. Live transitions allow describing finite and infinite sequence of events adjusted to conditions with statements for initial states. Possible trajectories are attained after exploring the results of initialized trajectories.

2.2.1.4 Hybrid Automata Description

As describe above, the semantics for an automata description requires a clear definition of the space (elements) present in the system. Let us assume a system where two hybrid automata are present. They are labeled as A1 and A2. Both automata interact using joint events. Transitions in the system are then considered and their consistency verified. Such verification is done by associative partial functions. To enable reachability, a state space is defined with a set of corresponding initial states. At this point, transition conditions are also described. Once the above elements are defined and verified, a parallel composition takes place. It is defined as the evaluation of both the consistency check versus the defined transitions.

2.2.1.5 Non-Deterministic Finite State Machine

It is important to highlight that in automata theory, nondeterminsitic finite state machines allow the automaton to change its state into several possible next states for a given input.

A non-deterministic finite-state automaton (NDFA) can be described as a 5-tuple $M \equiv (S, I, f, S_0, F)$, where its definition can be found in Table

Variable	Description
S	Finite set of *states*
I	Finite *input alphabet*
$f : S \times I \rightarrow P(S)$	*Transition Function* from each state-input pair to *a set of states*
S_0	*Initial state*
$F \subseteq S$	*Final States*

Table 2.3: Hybrid automata - Non-deterministic Finite-State Automata

2.3. [22] [42] Efforts for a hardware implementation are been described in [64].

2.2.2 Types of Automata

Rectangular Automata use the following characteristics to provide a different perspective in terms of descriptive means for implementation of models. In rectangular automata, the systems are constrained by providing a frame where the first derivative of each variable has a defined set of fixed values (range) for every control mode. The range of the variables does not affect the control scheme. Control switches define the operation to be performed over the variables; the variables can either remain unchanged or be changed to a set of fixed possibilities. *Multi-rectangular automata* provide means to vary flow conditions with respect to control switches. *Triangular automata* allow for a comparison of variables since each vertex is evaluated against its immediate neighbor.

2.2.3 Verification of Automata Traces

There are four analyses that can be made around the traces of the automata, H, which were initially defined as part of the problem description. The main questions are set as: reachability, emptiness, timed trace inclusion problem and time-abstract trace inclusion problem.[46] *Reachability* refers to the evaluation of a given trajectory (described as a Labeled Transition System), where an H automata visits a specific state given a by a set of operations in a well-defined control mode and an initialized trajectory. *Emptiness* can be interpreted as the evaluation of divergent initialized trajectories for the Labeled Transition System. *The time trace inclusion problem* evaluates if every timed trace of say H1 automata can be traced back to H2. *The time-abstract trace inclusion problem* evaluates the same mutuality between H1 and H2 automata but looks at it from the time-abstract traces perspective.

2.2.4 Basic Examples of Hybrid Automata

To illustrate in a better manner how this process is done an example for a railroad gate control is described here. There are three immediate automaton descriptions to be made: train, gate and controller automaton.

2.2.4.1 Train Automation

The example of a hybrid automaton for a train running on a circular track with 5000 m of circumference is described as follows. The system have a gate at 2000 m. The distance measured between the train and the gate

Variable	Definition	Notes
Shape of the track	Circular	
Circumference's length	2000 to 5000	Measured in meters
x	distance to the gate	Measured from the train itself in meters
$\frac{dx}{dt} = \dot{x}$	speed	Between 40 to 50. Measured in meters/second

Table 2.4: Hybrid automata - Train automation - Description

is described by the variable x. Figure 2.3 presents a model of the problem. Table 2.4 presents a simplified description of the train automation with a list of variables and their respective definitions. Table 2.5 describes the conditions and actions for the problem.

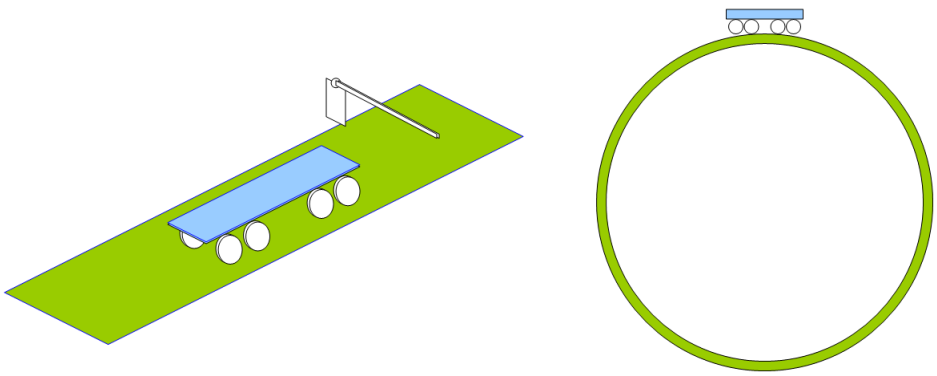

Figure 2.3: Hybrid Automata for a Train & Gate system automation. (a) Gate setup. (b) Circular Track

Gate Automaton The gate automaton will move to a maximum angle of 90°with a speed of 9°per second. There are four defined states: Open, Close,

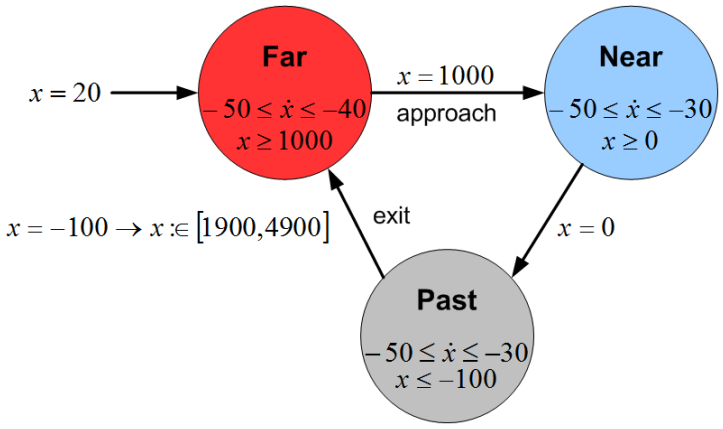

Figure 2.4: Hybrid Automata for a Train & Gate system automation[27]

Conditions	States and Actions
If $x = 100$ meters	State: Train is approaching gate. Reaction: Slow down to 30 m/s
If $x = -100$ meters	State: Train has passed the gate by 100m. Reaction: Exit event

Table 2.5: Hybrid automata - Train automation - States & Transitions

Move Up and Move Down. Two well defined variables speed and angular position are the main elements in the control for the change of states.

Gate Controller Automaton Controller automaton requires two main variables. The first one is related to the reaction delay of the controller ($\boldsymbol{u}$) and the second variable ($\boldsymbol{z}$) measures the time used in every operation. There are three well defined states which directly depend on the above mentioned variables and its derivatives. When the time changes the $\dot{z}$ variable is set to $\boldsymbol{1}$. If the reaction delay of the controller, $\dot{u}$, appears to be zero, it means

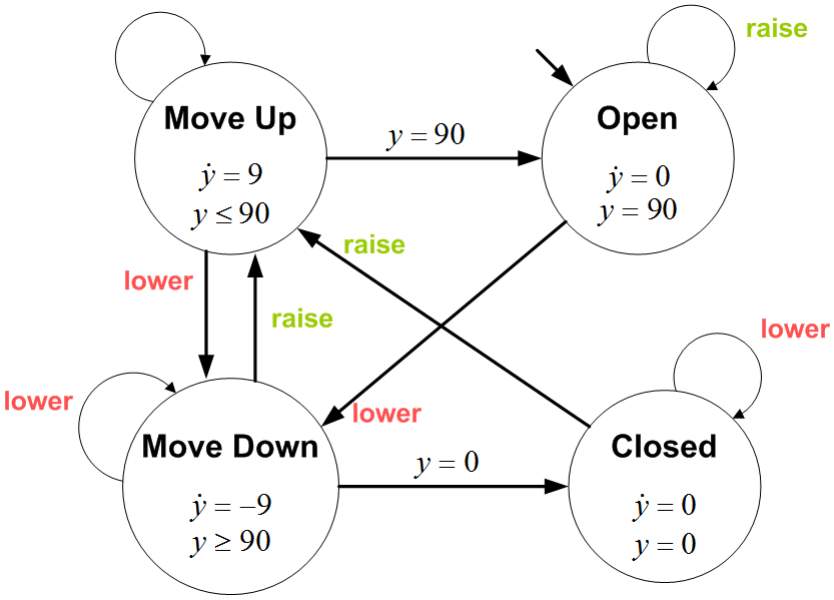

Figure 2.5: Gate automaton[27]

that there is a constant reaction delay at the controller level.

2.2.4.2 Electrical Circuits with Discrete Components

Buck converter without control feedback The following example uses a description for a buck converter for a variable load. The buck converter uses S1 as a switch timed to be ON for 6 seconds and to remain OFF for 4

Variable	Definition
y	Position of gate measured in degrees Max. 90
$\frac{dx}{dt} = \dot{x}$	Speed measured in degrees/seconds Max. 9

Table 2.6: Hybrid automata - Train automation - Gate Automaton

Variable	Definition
u	Reaction delay of controller
z	Clock for measuring elapse time

Table 2.7: Hybrid automata - Train automation - Gate Controller

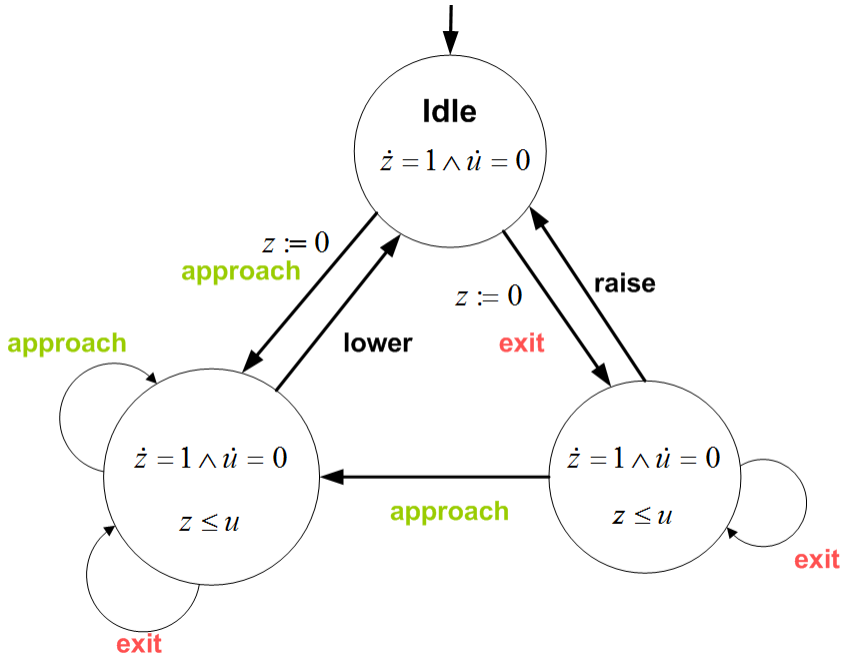

Figure 2.6: Hybrid automata - Train automation - Gate Controller[27]

seconds. The loads are selected by manipulating switch S2 every 4 seconds. A simple description using VHDL-ASM is shown in Appendix B for comparison purposes.

Buck converter with control feedback The main control loop is based on solid-state devices, mainly MOS transistors. The driving signals (gate voltage) are provided by a feedback setup that allows measuring hysteresis based on a

Discrete States	Condition
State A	Switch 1 → ON Switch 2→ Set to 1st load (R1)
State B	Switch 1 → ON Switch 2 → Set to 2nd load (R2)
State C	Switch 1 → OFF Switch 2 → Set to 2nd load (R2)
State D	Switch 1 → OFF Switch 2 → Set to 1st load (R1)

Table 2.8: Hybrid automata - Buck Converter - Description

Type of Var.	State Variables
Continuous	$i_L \rightarrow$ current through inductor $v_C \rightarrow$ Voltage across capacitor
Discrete	S1 → duration of ON/OFF state of Switch 1 S2 → duration of connection of Switch 2 to either R1 or R2

Table 2.9: Hybrid automata - Buck Converter - States & Variables

Clock variable	State	Notes
$\frac{dS1}{dt} = \dot{S1}$	1	Established for all states since clock reaches max. and min. for every change
$\frac{dS2}{dt} = \dot{S2}$	1	

Table 2.10: Hybrid automata - Buck Converter - Switches behavior

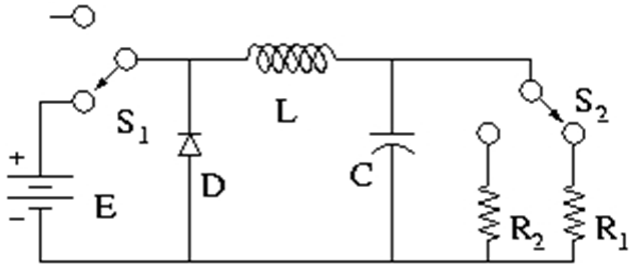

Figure 2.7: Hybrid Automata for a Buck Converter with multiple loads and no control feedback

reference voltage as shown in Figure 2.9.

The response of a buck converter can be seen as an underdamped second-order system.[32] Some of the control feedback blocks required for a fully operational setup are: zero pulse generator, mono-shot pulse generator, startup pulse generator, zero crossing detector, MOS switches drivers and hysteresis comparator.

2.3 Petri Nets

Petri Nets are defined as a mathematical modeling language used to described distributed systems. Those systems can have dynamic concurrency behaviors with discrete flows. A Petri Net consists of three types of elements. The first elements, *bars*, are used to described transitions. The second ones, *circles*, represent conditions (some times called places) and the third set of elements, *arcs*, connect the above mentioned items.

The connection elements are noted by *arcs* allowing the clear distinction between pre and post conditions. Arcs are usually represented with arrows.

Discrete States	Condition	Action	Reset
State A	PMOS $\rightarrow$ ON NMOS $\rightarrow$ OFF PMOS Control $\rightarrow$ ON NMOS Control $\rightarrow$ OFF	$\frac{dV_{out}}{dt} = \dot{V_{out}} = 1$ or $\frac{dV_{CX}}{dt} = \dot{V_{CX}} = -1$	$V_{CX} = E$
State B	PMOS $\rightarrow$ ON NMOS $\rightarrow$ OFF PMOS Control $\rightarrow$ OFF NMOS Control $\rightarrow$ ON	$\frac{dV_{out}}{dt} = \dot{V_{out}} = 1$ or $\frac{dV_{CX}}{dt} = \dot{V_{CX}} = -1$	$V_{CX} = E$
State C	PMOS $\rightarrow$ OFF NMOS $\rightarrow$ ON PMOS Control $\rightarrow$ OFF NMOS Control $\rightarrow$ ON	$\frac{dV_{out}}{dt} = \dot{V_{out}} = -1$ or $\frac{dV_{CX}}{dt} = \dot{V_{CX}} = 1$	$V_{CX} = V_k$
State D	PMOS $\rightarrow$ OFF NMOS $\rightarrow$ ON PMOS Control $\rightarrow$ ON NMOS Control $\rightarrow$ OFF	$\frac{dV_{out}}{dt} = \dot{V_{out}} = -1$ or $\frac{dV_{CX}}{dt} = \dot{V_{CX}} = 1$	$V_{CX} = V_k$

Table 2.11: Hybrid automata - Buck Converter with FB control - Description

Initial State	Final State	Condition
State A	State B	$T \geq T_{on}$
State B	State C	PMOS Control $\rightarrow$ OFF NMOS Control $\rightarrow$ ON
State C	State D	$V_{CX} = 0$ $V_{out} \leq V_{lth}$

Table 2.12: Hybrid automata - Buck Converter with FB control - States & Variables

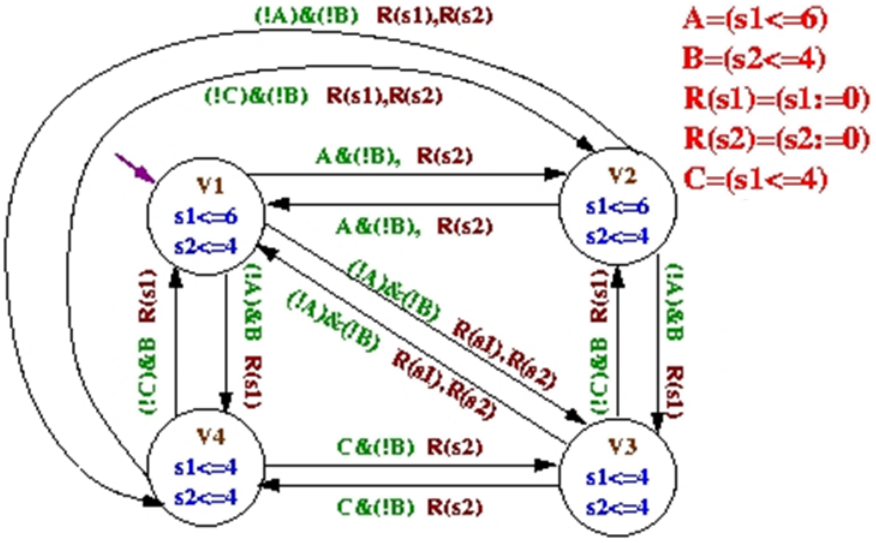

Figure 2.8: Hybrid Automata States for a Buck Converter without control feedback and multiple loads

The configuration of the net is represented using *tokens*, which are activated (move from place to place using transitions) in a single unique operation. [18][45]. Table 2.13 shows a general representation of the elements present in most Petri Net systems.

Figure 2.10 shows a simple example with two places (p_1, p_2) and one transaction (t_1). The input place is represented as p_1 and the output place as p_2. The state transition has the form $(1,0) \rightarrow (0,1)$. The transition node can be activated only if there is at least one token at each of its input places.

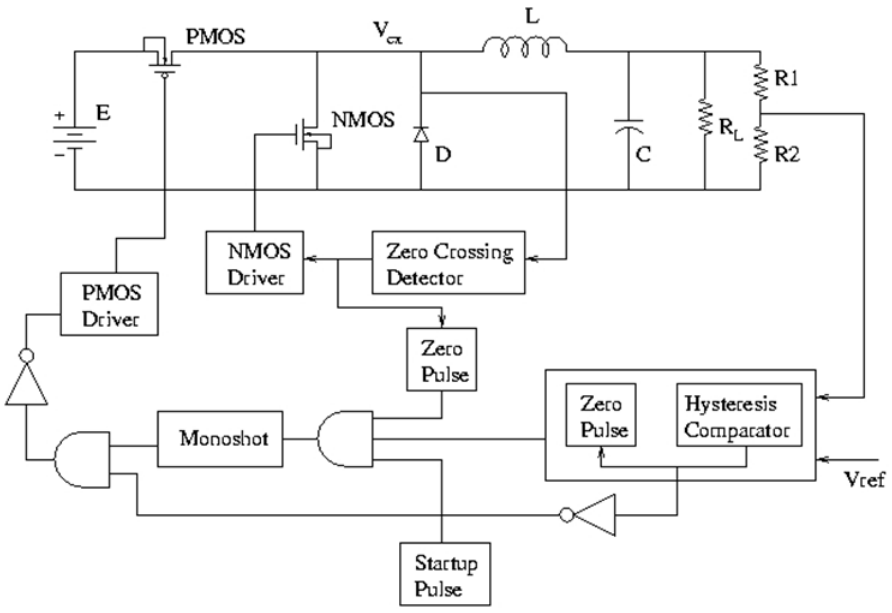

Figure 2.9: Hybrid Automata States for a Buck Converter with control feedback

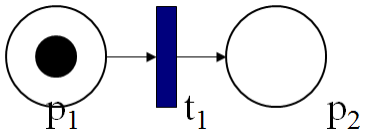

Figure 2.10: Petri Net for a two state and one transition system

2.3.1 Properties of Petri Nets

2.3.1.1 Sequential Execution

Figure 2.11 shows a Petri Net with three states (p_1, p_2, p_3) and two transitions (t_1, t_2). Transition t_2 can only take place if transition t_1 takes place previously. As a consequence t_2 is sequentially linked to t_1

General description	Individual description	Formulation
$C = (P, T, I, O)$	Places	$P = \{p_1, p_2, p_3, \cdots, p_n\}$
	Transitions	$T = \{t_1, t_2, t_3, \cdots, t_n\}$
	Input	$I : T \rightarrow P^r$ where r is the number of places
	Output	$O : T \rightarrow P^q$ where q is the number of places
μ	Tokens (associated to places in the Petri Net)	$\mu = \{\mu_1, \mu_2, \mu_3, \cdots, \mu_n\}$

Table 2.13: Petri Nets - General representation

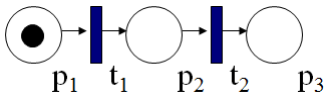

Figure 2.11: Petri Net - Properties - Sequential execution

2.3.1.2 Synchronization

Whenever there are two tokens at concurrent states, a transition can be enabled. In Figure 2.12, each place has a token, and as such, transition t_1 can be fired.

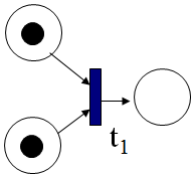

Figure 2.12: Petri Net - Properties - Synchronization

2.3.1.3 Merging

In the case shown in Figure 2.12, there were two concurrent tokens. If their respective *arcs* arrive to the same transition, then it is possible to have a merging scenario as shown in Figure 2.13.

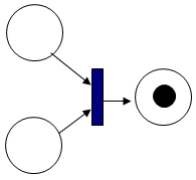

Figure 2.13: Petri Net - Properties - Merging

2.3.1.4 Concurrency

Whenever two tokens are set in places that are directly preceded by a transition, concurrency can be defined. It allows to model processes with distributed control. In Figure 2.14, there are two states with unique tokens and both are linked to a previous similar transition stage. As a consequence, transitions t_1 and t_2 will take place concurrently.

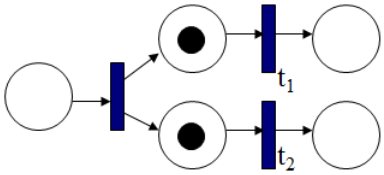

Figure 2.14: Petri Net - Properties - Concurrency

2.3.1.5 Conflict

Conflict between transitions can take place if more than one arc is connected to a single transition having as origin a singular place as shown in Figure 2.15. The three initial places can fire transitions t_1 and t_2 at any time since each has an unique token. However, as a consequence of such setup, multiple places of origin cause one token to be left behind. Such behavior can be controlled by assigning probabilistic (non-deterministic) weights to provide a more controllable net.

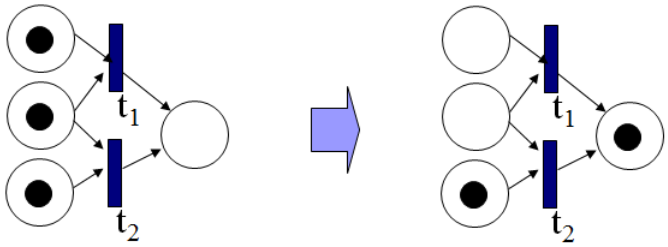

Figure 2.15: Petri Net - Properties - Conflict

2.3.2 Petri Net Example

2.3.2.1 Restaurant Service

As a simple example, Figure 2.16 presents a Petri Net for a restaurant service where there are two customers and just one free waiter (each represented as a *place*). The final places for the tokens are set as labeled as *eating*.

Without the *merging* property, transition from the initial places (Customer 1, Customer 2, and Free Waiter) will be done without any order. In this case, either Customer has to wait until a free Waiter moves to the "Take

Order" transition. From there, the flow towards the final place (eating) is started. Notice that the Petri Net offers a specific order of actions aligned according to the initial places. As such, neither Customer will go to the "Order Taken" place.

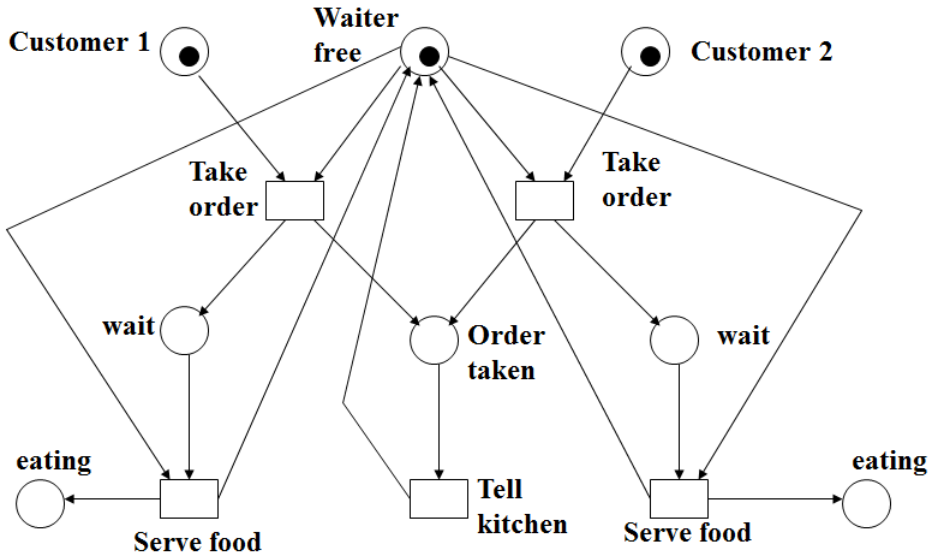

Figure 2.16: Petri Net - Example - Restaurant service

2.3.2.2 Ideal Diode

Figure 2.17 shows a representation of an Ideal Diode using a Petri Net. A token is set in the *ON* state, and it is fired whenever the current flowing through the diode is less than zero ($I_{diode} < 0$). The Petri Net will then move the token to the *OFF* state.

As a second sequence, whenever the token is present at the *OFF* state it will be fired if the Diode's Voltage is greater than zero ($V_{diode}¿0$). The Petri

Net will then have the token moved to the *ON* state.

It is important to highlight the fact that *Finite State Machines* formalisms are a subset of the Petri Nets representations with just one input and a single output. Finite State Machines as a subset of Petri Nets allow just one token at the initial place[45]

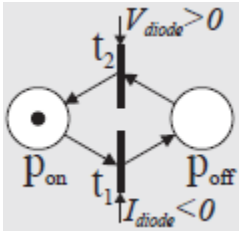

Figure 2.17: Petri Net - Example - Ideal Diode [45]

2.3.3 Hierarchical Petri Nets

Petri Nets require an extra framework to set up components hierarchy. To provide a hierarchical structure Petri Nets require two main chains of flow. One related to the *interface*, which is external, and one set as a realization, which is *internal* [57]. In order to implement a hierarchical semantic setup, states and events will have to clearly been delimited.

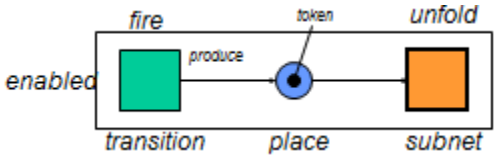

Figure 2.18: Petri Net - Hierarchical - Component representation

Petri Net systems are usually described as systems with places being set

as inner entities which cannot be accessed by any external actor as shown in Figure 2.18. Figure 2.19 describes a Petri Net with unfolded inner blocks. Once the components are unfolded, it is possible to create a sequence with actions that directly influence the way tokens are moved through the transition stages as shown in Figure 2.20. A complete description of the system will require adding inner elements as shown in Figure 2.21

Compositionality uses semantics to determine the behavior of components in any context. It as well allows mapping components to mathematical objects and it is only done for interfaces. Many approaches to the validation problem have been adopted.

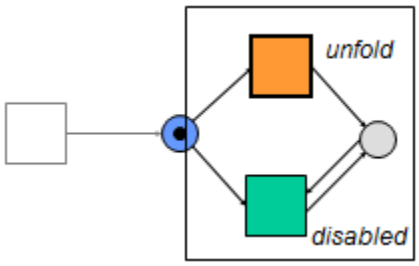

Figure 2.19: Petri Net - Hierarchical - Component representation with unfolding [57]

Using *Net Components* allows describing any inner components and their connections. The semantics of the Net Components use a specific set of event types which are: external consumption, firing, and external production.

A simple description for the Net Component semantics can be seen in Equation 2.1, where a Petri Net with a resulting initial state X, consumes c, fires f, and by producing p results in a new X' state

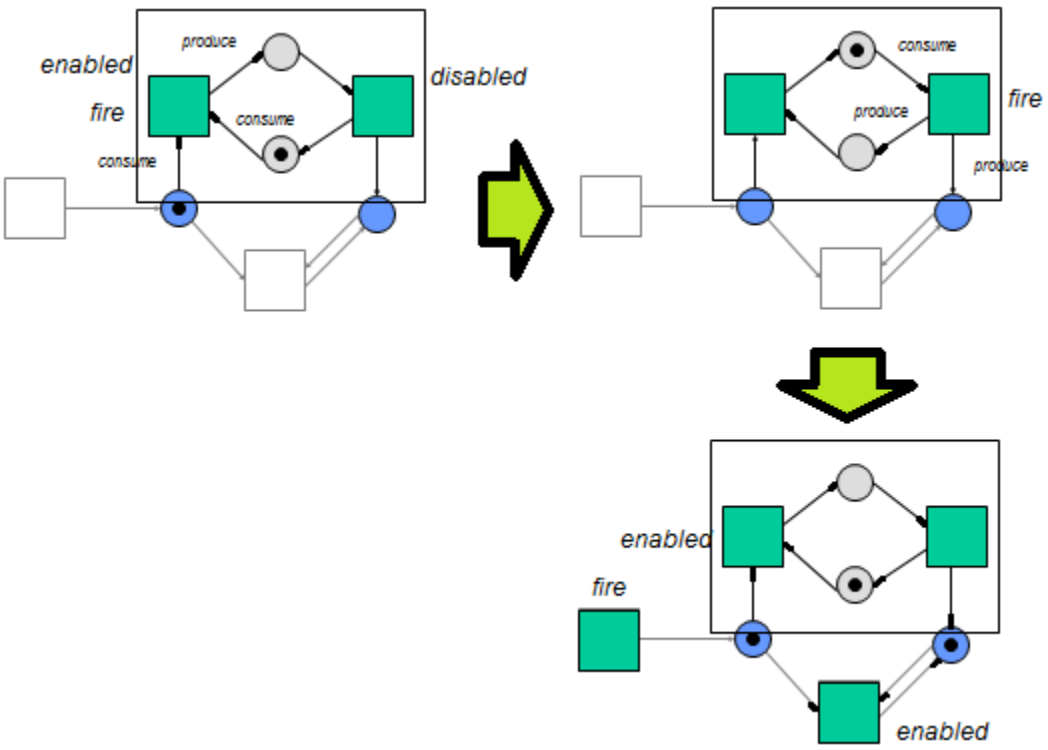

Figure 2.20: Petri Net - Hierarchical - Component representation with unfolding - Sequence with enable actions [57]

$$X \xrightarrow{c,f,p} X' \tag{2.1}$$

There is always the chance of having a possible null case where the final event is the initial event, which can be described as $X \xrightarrow{0,0,0} X'$

As shown in [37], adding time properties to Petri Nets can be used as a framework for verification of Analog and Mixed-Signal systems. The complexity of such systems can greatly be simplified using enhanced descriptions for flow interaction, which requires establishing *Transition Systems*. Transi-

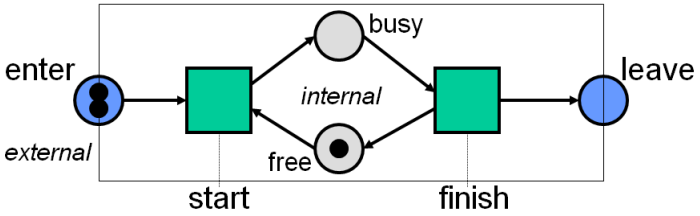

Figure 2.21: Petri Net - Hierarchical - Component representation with Internal Blocks [57]

tion Systems provide the required framework needed for a description using directed graphs where *Nodes* describe states and *events* provide a representation of labeled *arcs*.

2.4 Bond Graphs

Bond Graphs are a domain-independent object-oriented representation of physical systems with a simplified description of their dynamic behaviors[6]. Bond Graphs provide the required framework for the creation of models where interaction take place in different physical domains. Such characteristic make them ideal vehicles for hybrid systems descriptions. A very important feature of bond graph is their non-causal characteristics, which allow having a response from the modeled system before any stimuli is provided to it. A simple example of a non-causal system is shown in Equation 2.2, where x is the input and y is the output for a given system.

$$y(n) = x(n+1) + x(n+2) + y(n+1) \tag{2.2}$$

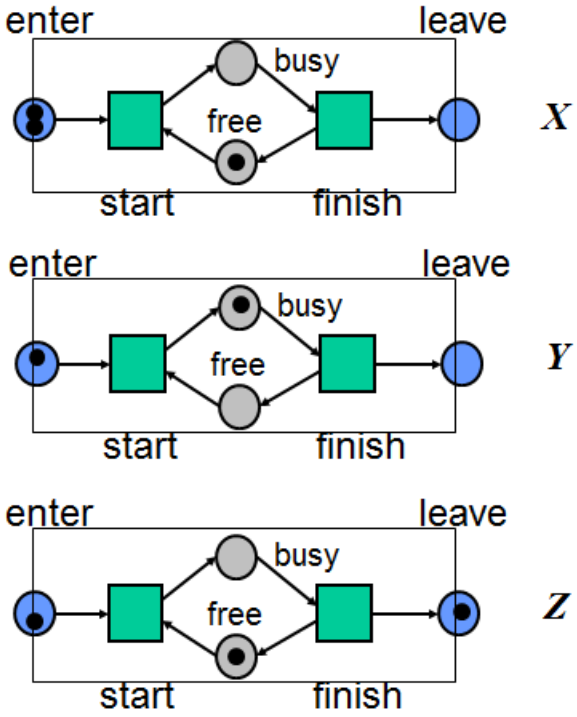

Figure 2.22: Petri Net - Hierarchical - Component representation with Internal Blocks - Semantics example [57]

The main components of Bond Graphs are vertices which are idealized descriptions of the physical system and as such can be assumed to be sub-models; the second components are the *edges* (some times called bonds) which represent an ideal connection between sub-model ports. Figure 2.23 shows the energy flow between two vertices using a single Bond.

Connections or Bonds have a duality characteristic in terms of direction. They can either represent power or computational causality. Such differentiation solves the issue related to sign-placing while describing the system, which is known as *casual analysis*. Figure 2.24 shows a description for a system with

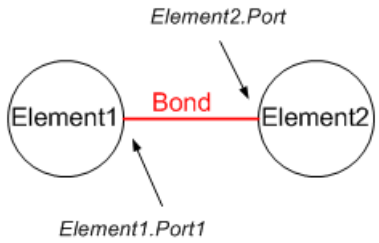

Figure 2.23: Bond Graphs - Energy Flow - Interaction of two submodels using a single *bond*

proper signal assignment. Notice, however, there is neither assumption nor description of the power direction in the Figure. The respective mathematical expressions for such signal assignment are described in Equation 2.3

Figure 2.24: Bond Graphs - Signal direction

$$\begin{aligned} element1.e &:= element2.e \\ element2.f &:= element1.f \end{aligned} \tag{2.3}$$

2.4.1 Bond Graph Example - RLC Serial

As a simple example, a RLC circuit will be discussed. Figure 2.25 shows a simple description of the circuit.

The mathematical expressions governing the elements of the systems can be seen in Equation 2.4. The electric elements present in the system can

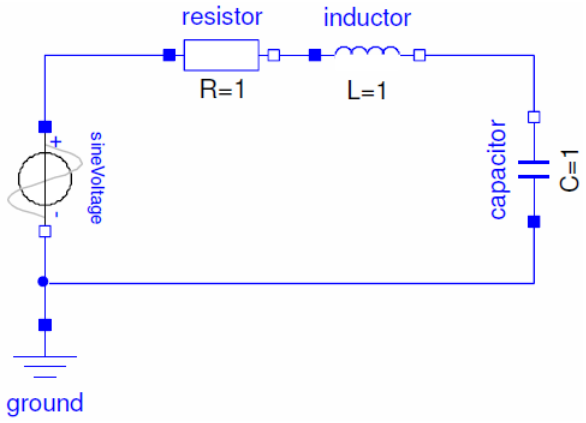

Figure 2.25: Bond Graphs - example - RLC circuit

then be described as functional blocks with power ports as shown in Figure 2.26

$$\begin{aligned} V_R &= iR \\ V_C &= \frac{1}{C}\int i\mathrm{d}t \\ i_L &= \frac{1}{L}\int v\mathrm{d}t \end{aligned} \tag{2.4}$$

After looking at Figure 2.25, it is possible to conclude that the current i is the same for all the elements in the system. However, their voltages are different. Following the above mentioned premise, it is possible to create a Bond Graph as shown in Figure 2.27. The common i flowing is change to be *1* which is usually called a 1-junction. Such representation assures that the flow (current) through all the components is the same and also guarantees that the sum of the efforts (voltage) is equal to zero (considering the restrictions im-

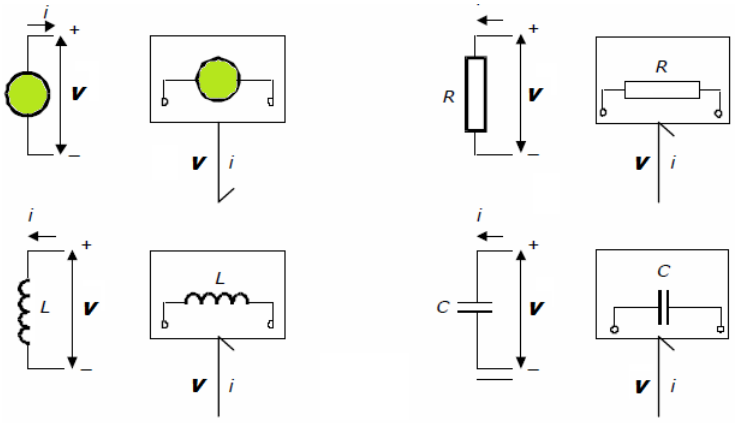

Figure 2.26: Bond Graphs - example - RLC serial circuit - Power ports [6]

posed by the sign assignment, which is directly linked to the sum of the power). The *half arrow* represents the direction of the power in the bond. Voltage for all the components is mapped to a domain-independent effort variable while the Currents are described by a domain-independent flow variable.

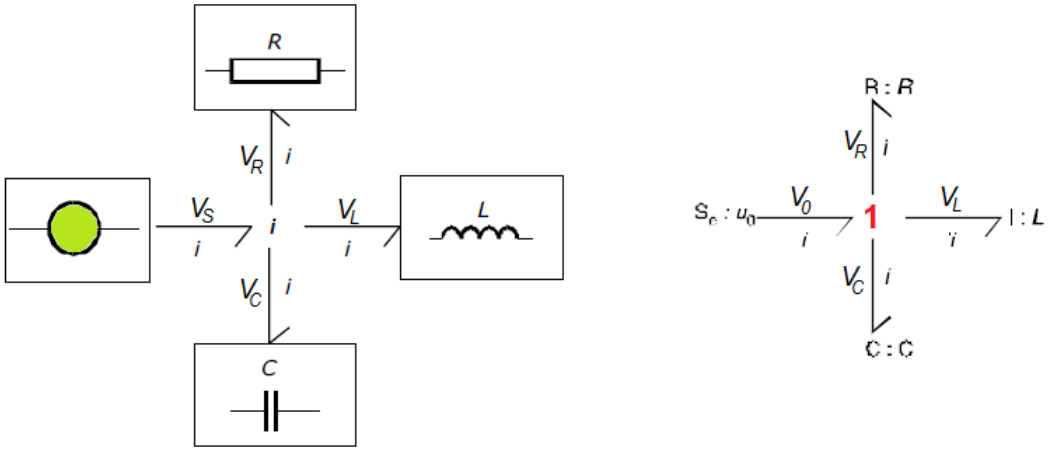

Figure 2.27: Bond Graphs - example - RLC serial circuit - Standard symbols representation [6]

2.4.2 Bond Graph Example - RLC Parallel

A similar analysis for the RLC parallel circuit can be carried out. However, since the voltage is the same for all the electrical components, the representation of it will have to be done with a 0-junction. In which case, the effort (voltage) over all the connected bonds is the same and the sum of flow (current) is equal to zero as shown in Figure 2.28. The sum of the flow has to be carried out considering the sign representation.

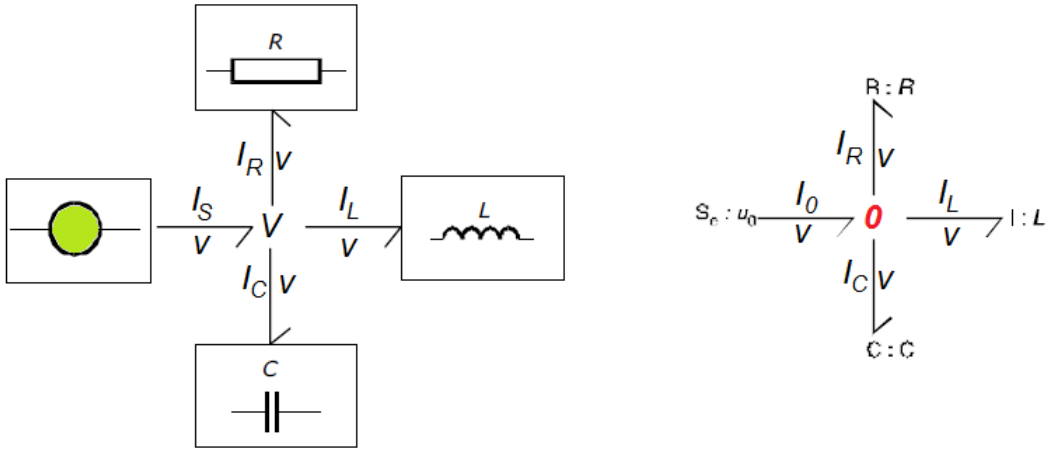

Figure 2.28: Bond Graphs - example - RLC parallel circuit - Standard symbols representation

2.5 Modeling Languages

At the beginning of this project, it was established as one of the main goals to look for open-source solutions as much as possible. A wide variety of simulation environments were considered. Table 2.14 shows some of them and presents a simple comparison.

Language	Source Code	Features	Notes
Verilog AMS	Not Open	Modular, easy description of electrical systems, intricate ODE implementations	Requires complex models for not electrical elements and ODE
Simulink	Not Open	Capable of ODE implementation, electrical descriptions depend on Matlab based modules	Implementations such as Si-COS can be considered as an open-source alternative
Modelica	Open	Capable of ODE implementations, easy integration of electrical models. Allows multi-domain simulation (electrical, mechanical, etc).	It does not have a direct hardware implementation

Table 2.14: Modeling & Simulation environments

It is important to highlight that the goal for this report was not to make a comprehensive comparison of the above mentioned languages. Modelica was chosen due to its integrated simulation and multi-domain capabilities.

2.5.1 Modelica

Modelica is an object-oriented language for modeling physical systems. It was initially developed by the Modelica Association[1] which is a non-governmental international association. There are two major commercial simulators for Modelica. The first one is Dymola, which was recently acquired by Dassault Systems[2] (CATIA developers). The second commercial simulator is MathModelica[3] from Wolfram (Mathematica developers). As an alternative to the above simulators there is an open-source simulator called Open-Modelica[4] which is the one selected for this work. Some of the main Modelica features are presented in Table 2.15

A model can be described using a very simple structure [28] as shown in the Listing 2.1 (For an extra set of Modelica examples, refer to Appendix A). Some of the main Modelica programming components are shown in Table 2.16. Notice the convention about capital letters.

[1] http://www.modelica.org
[2] http://www.3ds.com/products/catia/portfolio/dymola
[3] http://www.mathcore.com/products/mathmodelica/
[4] http://www.openmodelica.org/

Feature	Advantages
Object-oriented	Scalable
Multi-Domain	Thanks to its large number of component libraries, it is possible to simulate electric circuits, mechanical systems, petri nets, etc)
Open-Source	Allow modification of the original libraries (can be done with the commercial version as well) and source of the main simulator to customize attempted simulation & verification goals.

Table 2.15: Modelica Features

Listing 2.1: Basic structure for Modelica scripts.

```
model ModelName
    // declaration of public variables
    // These variables can be accessed as inputs or outputs
    // They can be called by using "ModelName.Variable var",
    // where "var" is the name of the variable

protected
    // used to create the object-oriented setup
    // declaration of objects is carried out
    // for internal (hidden) variable (state)

equation
    // algebraic and differential equations

end ModelName
```

Type	Keyword	Notes
Predefined Types	Real	Real number
	Integer	Integer
	Boolean	True or False
Type modifiers	parameter	Element does not chance during the simulation. It can be initialized
	constant	Fixed element
	discrete	Constant variable (for a discrete state)
	flow	Allows representation of conservation flow. By convention positive means flow enters component
equation Funct./Vars.	der	Derivative of *real* value
	pre	Left-limit of *discrete* value
	edge	True when *discrete* variable is discontinuous
	time	simulation time
equation Commands	reinit	Reset variable
	connect	Connects variables
	when	Executes sentence(s) when its argument is true

Table 2.16: Modelica Components [28]

Chapter 3

Electric Systems Modeling

There are several constraints related to the way circuit modeling is carried out. As it is currently being done several simplifications are made to secure an efficient and fast analysis. Most of the theoretical models use the Kirchhoff's expressions for current and voltage. However, extreme simplification can produce inaccurate results when one wants to simulate circuits with configurations different than lumped arrangements. Mathematical modeling of lumped systems is conditioned to well-defined assumptions that encompass some of Kirchhoff's laws.[9] Lumped MOS models present several limitations for fast transient signals (RF and microwave) as shown in [4]. Such limitations reduce the models accuracy and require assumptions such as quasi static (QS) behaviors. It works well for wavelengths greater than a quarter wavelength.

3.1 Elements of RF and Microwave Systems

Given the nature of the modern micro-systems where the wavelength is usually smaller than the circuits characteristic length, a distributed element model with dynamic behaviors based on Maxwell's equation should be preferred. Comparisons between lumped element and distributed mod-

els have been carried out especially for systems with non-linear transmission elements.[43]

As a simple example, let us assume a microsystem expected to work at 60 GHz.[53] The wavelength for such system can then be calculated as shown below.

$$\lambda = \frac{c}{f} = \frac{2.999 \times 10^8}{60 \times 10^9} \approx 5.0 \times 10^{-3} m = 5.0mm \tag{3.1}$$

The size of the microsystem is approximately the same as the wavelength of the attempted specifications. In such case the only way to have an accurate model requires a distribute impedance elements setup.

Distributed impedance circuits assume that voltage and current waves travel across conductors in a propagation mode since there is a not instantaneous change due to ideal wires, which requires understanding the properties of the conductors impedance. The main assumption for the distributed model presents a continuous distribution of impedance elements throughout the transmission line.[36]

$$\bigtriangledown \cdot \mu_0 H = 0 \tag{3.2}$$

$$\bigtriangledown \cdot \varepsilon_0 E = \rho \tag{3.3}$$

$$\bigtriangledown \times H = J + \varepsilon_0 \frac{\partial E}{t} \tag{3.4}$$

$$\bigtriangledown \times E = -\mu_0 \frac{\partial H}{t} \tag{3.5}$$

There are some other limitations associated when MOS devices are to be modeled as shown in [4]. Alternatives where the transient and steady state simulations are used simultaneously to characterize a microsystem have been proved to be acceptable alternatives as shown in [15]. However, the same problem is seen since they require multi-domain descriptions for some of the components which limit the strategy to be followed .

At small dimensions, it is possible to include the transmission line model as part of the distributed circuit. Kirchhoff's laws can then be applied to differential elements which are assumed to have infinitesimal length and as such instantaneous effects can still be attained. The transmission model considers not only the ohmic losses in the conductor but the ones created as part of the differential. If one assumes a macro-model with discrete components (e.g. PCB board design), it is a common design practice to have the conductors carrying the main signals travel across a set of ground planes. Such sandwich configuration allows isolation from external noise and provides a better shield for EMI. The same principles are applied to the IC design realm and as such considerations for dielectric elements have to be made.

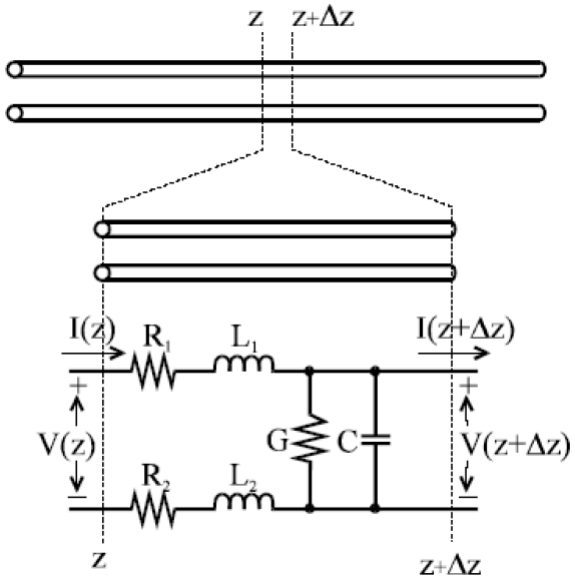

Figure 3.1: Model for a transmission line

An expression for the transmission line model shown above can then be implemented as follows.

$$\frac{\partial i(t,z)}{\partial z} = -\left(Gv(z,t) + C\frac{\partial v(z,t)}{\partial t}\right) \tag{3.6}$$

$$\frac{\partial v(t,z)}{\partial z} = -\left(Ri(z,t) + L\frac{\partial i(z,t)}{\partial t}\right) \tag{3.7}$$

$$\frac{\mathrm{d}V(z)}{\mathrm{d}x} = -(R + j\omega L)I(z) \tag{3.8}$$

$$\frac{\mathrm{d}I(z)}{\mathrm{d}x} = -(G + j\omega C)V(z) \tag{3.9}$$

Transmission lines on thin substrates are studied in [54].

$$\gamma = \alpha + j\beta = \sqrt{(R + j\omega L)(j\omega C)} \tag{3.10}$$

$$Z_0 = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \tag{3.11}$$

$$v(z,t) = \left|V_0^+\right| e^{-\alpha z} cos(\omega t - \beta z + \theta^+) + \left|V_0^-\right| e^{-\alpha z} cos(\omega t + \beta z + \theta^-) \tag{3.12}$$

Models for the above mentioned elements can be implemented using several techniques. As shown previously in Section 2.3.2.2, a simple example for an Ideal Diode is carried out using *Petri Nets*. A complete system description can be done using the methodology proposed in [45]. It consists of three basic steps as shown in the following list:

1. *Collect* the equations of all components of the system
2. Add trivial equations of the form $a = b$ for every component *connection*.
3. *Sort* all equations into an explicit forward sequence

3.2 Circuits Descriptions Using Modeling Languages

Examples of the use of high-level models have been implemented using languages such as Modelica [12]

3.2.1 Simple Circuit Description Using Bond Graphs

Using the descriptions done in Sections 2.4.1 and 2.4.2 as a reference, it is possible to describe a model for a basic circuit using a simple Bond Graph as shown in Listing 3.1. [10] Figure 3.2 shows a representation of the circuit using a Bond Graph description.

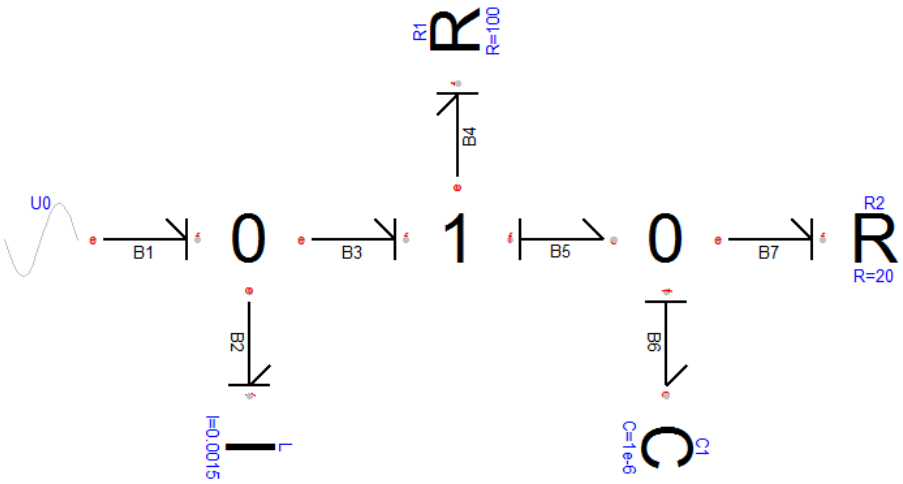

Figure 3.2: Simple Circuit description using Bond Graphs

3.2.2 RLC Serial Circuit

A simple RLC circuit (Figure 2.25) is implemented using Bond Graphs as shown in Listing 3.2. The model uses the BondGraphs Modelica library for the representation of the components.[10] The resulting signals are plotted in Figure 3.3.

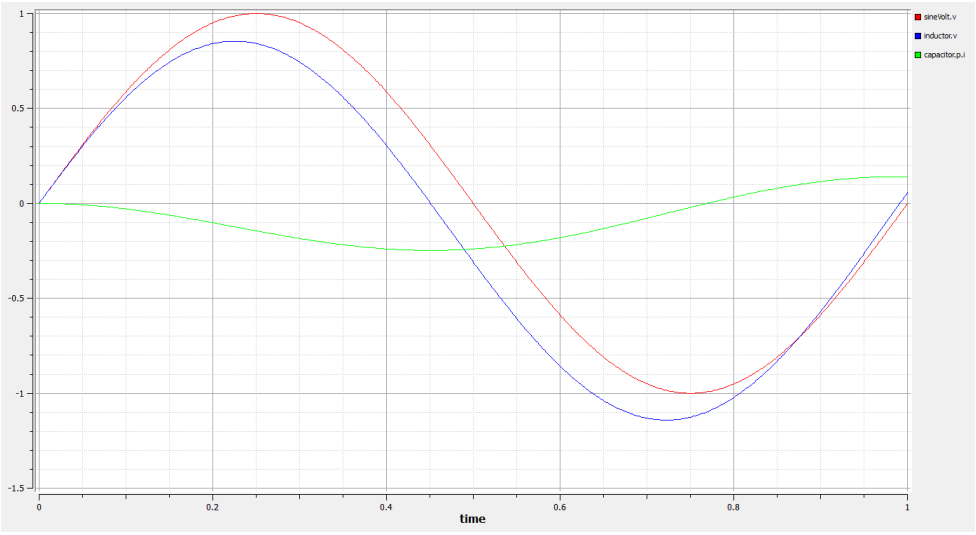

Figure 3.3: RLC circuit - Signals after simulation using Modelica BondLib library

3.2.3 Ideal Conventional Diode

The simplest example one can use to understand the interaction between continuous and event driven equations is the ideal diode.[19] Listing 3.3 shows a simple Modelica script to implement a simple model. The behavior of an ideal diode can be represented as shown in Equation 3.13.

$$i(t) = I_s[\exp(\alpha v(t)) - 1] \tag{3.13}$$

3.2.4 Esaki (Tunnel) Diode

A tunnel diode is a p-n junction device that presents *negative resistance*. [33][58] It only takes place at small voltages, though. The total current I in a tunnel diode can be expressed as shown in Equation 3.14

$$I = I_{tun} + I_{diode} + I_{excess} \tag{3.14}$$

For the case of the *tunnel current* Equation 3.15 presents a mathematical description. The maximum negative resistance can be found as shown in Equation 3.16 and the peak voltage using Equation 3.17.

$$I_{tun} = \frac{V}{R_0} \exp\left[-\left(\frac{V}{V_0}\right)^m\right] \tag{3.15}$$

$$|R_{d\max}| = R_0 \frac{\exp\left(\frac{1+m}{m}\right)}{m} \tag{3.16}$$

$$V_p = \left(\frac{1}{m}\right)^{\frac{1}{m}} V_0 \tag{3.17}$$

The *excess current* which is an additional tunnel current directly linked to parasitic tunneling due to impurities determines the minimum valley current as shown in Figure 3.4. It can be described as presented in Equation 3.18.

$$I_{excess} = \frac{V}{R_v} \exp\left[\left(\frac{V - V_v}{V_{ex}}\right)\right] \tag{3.18}$$

A functional Modelica script for the model can be seen in Listing 3.4.[13] For a complete model (including icon and documentation) go to Appendix A. A schematic representation of the circuit used for the characterization can be seen in Figure 3.5 and its representation as a Modelica script in Listing 3.5.

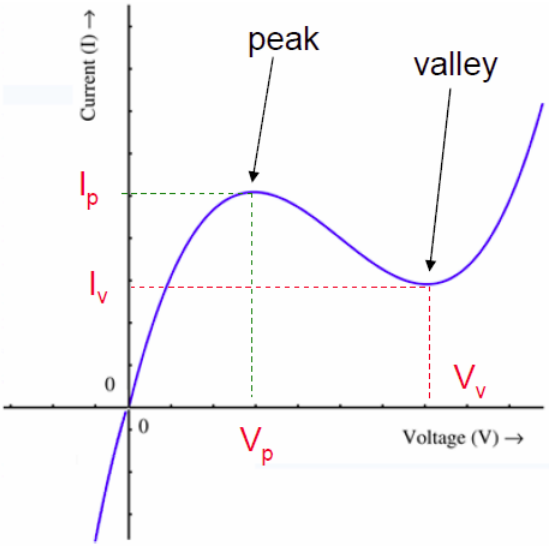

Figure 3.4: Ideal Tunnel Diode - Equation extraction model

Figure 3.6 shows a comparison between an ideal diode and its tunnel diode counterpart.

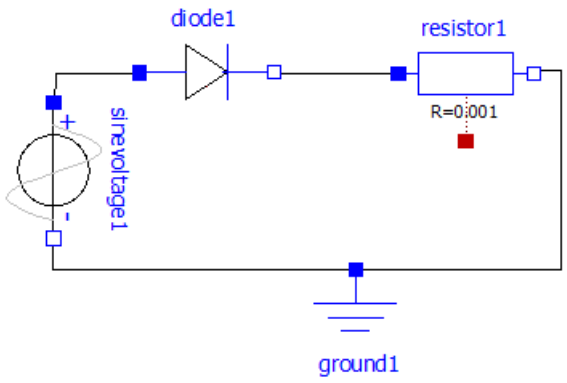

Figure 3.5: Modelica description for Ideal Tunnel diode characterization

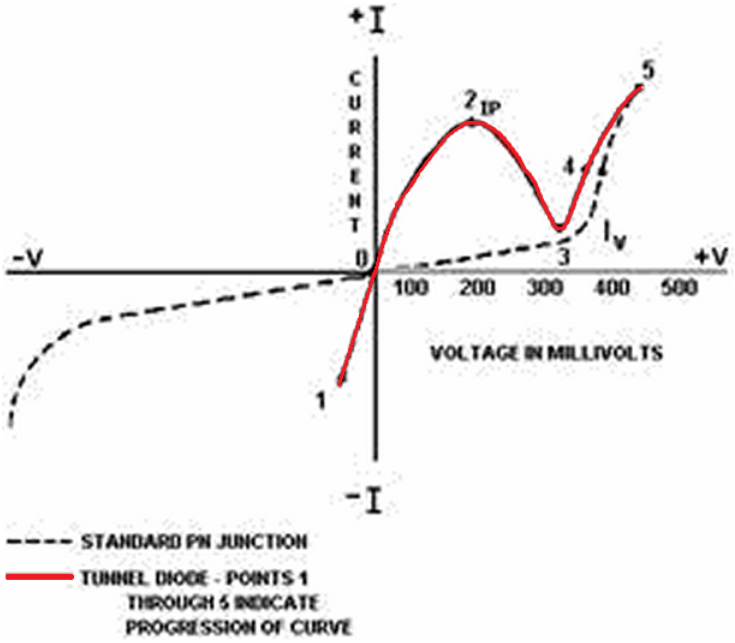

Figure 3.6: Voltage versus Current for the Esaki (tunnel) diode [47]

3.2.4.1 Tunnel Diode Oscillator

A simple model used to learn about the oscillatory properties of the Tunnel Diodes can be seen in Listing 3.6. The Tunnel Diode oscillates whenever it is fed with a voltage near the estimated threshold, which in this case is *300mA*. Some details about its fabrication and operational issues are described in Appendix C.

3.2.4.2 Petri Net Representation of Tunnel Diode

Figure 3.8 shows a simple representation for a Tunnel Diode using a Petri Net. Some examples of complex AMS systems modeled using Petri Nets can be found in [38] which ratifies the possibility of using such approach.

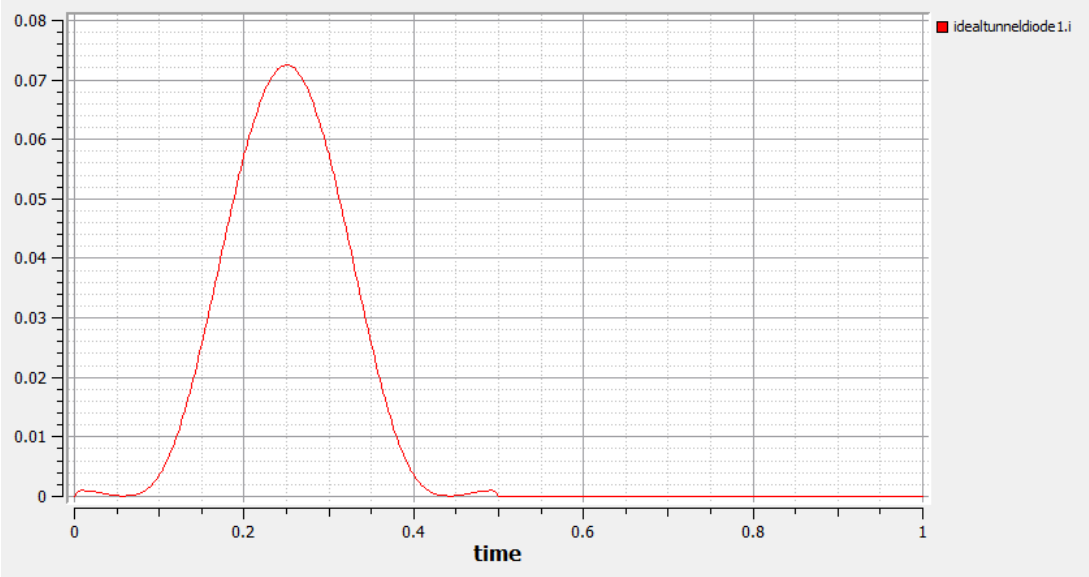

Figure 3.7: Current at the Ideal Tunnel diode [Time VS Current]

There are four well defined places: *OFF*, *Cutt-Off*, *Negative Resistance*, and *Saturated*.

The *OFF* state is set with the assumption that there is not conduction whenever the voltage is less than zero. Such a premise is made in order to simplify the model. The *Cut-Off* state is defined between 0 and V_{peak}. There is negligible electrical conduction at this stage. The *Negative Resistance* state is defined between V_{peak} and V_{valley}. This stage presents a very interesting quantum property that allows the presence of Negative Resistance behavior for small voltages. Devices made out of Germanium present V_{valley} voltages close to *300mA*. The *Saturated* state takes place whenever the Voltage is set to be greater than V_{peak}. At this point the diode behaves as a normal forward current device.

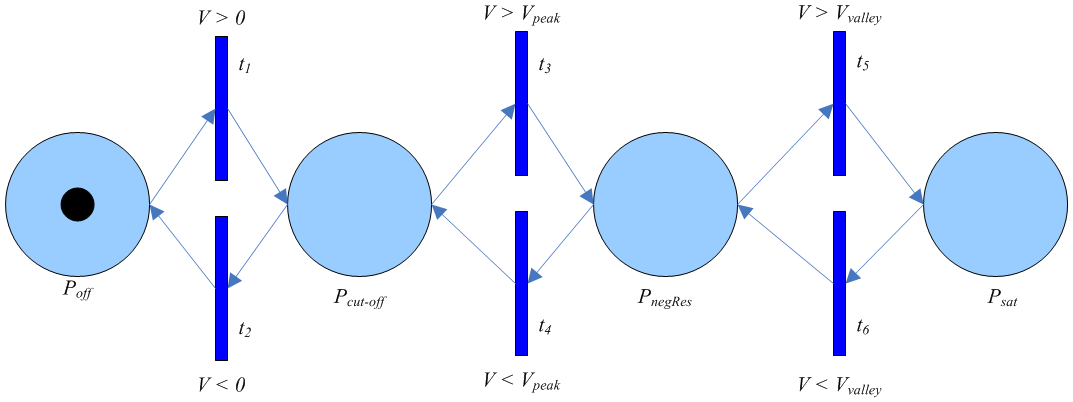

Figure 3.8: Petri Net - Representation of Tunnel Diode

To represent the Petri Net using Modelica the Extended-Petri Net library was used. [17] Some modifications required to be done since such library was designed for Dymola environments, which is a commercially licensed software, and it is not part of the original Open-Modelica environment. Listing 3.7 presents a basic Modelica representation for the Petri Net shown in Figure 3.8.

Listing 3.1: Simple Bond Graph

```
model SimpleBondGraph "A simple bond graph of an electrical ckt"
  constant Real pi = 4 * Modelica.Math.atan(1);

  BondLib.Sources.sinSe U0(e0 = 10, freqHz = 2500 / pi) ;
  BondLib.Bonds.fBond B1;
  BondLib.Junctions.J0p3 J0p3_1;
  BondLib.Bonds.fBond B2;
  BondLib.Passive.I L(I = 0.0015, f(start = 2));
  BondLib.Bonds.fBond B3;
  BondLib.Junctions.J1p3 J1p3_1;
  BondLib.Bonds.fBond B4;
  BondLib.Passive.R R1(R = 100);
  BondLib.Bonds.eBond B5;
  BondLib.Junctions.J0p3 J0p3_2;
  BondLib.Bonds.eBond B6;
  BondLib.Passive.C C1(C = 1e-06, e(start = 1));
  BondLib.Bonds.fBond B7;
  BondLib.Passive.R R2(R = 20);

equation
  connect(U0.BondCon1,B1.eBondCon1);
  connect(B1.fBondCon1,J0p3_1.BondCon1);
  connect(J0p3_1.BondCon3,B2.eBondCon1);
  connect(B2.fBondCon1,L.BondCon1);
  connect(J0p3_1.BondCon2,B3.eBondCon1);
  connect(B3.fBondCon1,J1p3_1.BondCon1);
  connect(B4.eBondCon1,J1p3_1.BondCon3);
  connect(R1.BondCon1,B4.fBondCon1);
  connect(J1p3_1.BondCon2,B5.fBondCon1);
  connect(B5.eBondCon1,J0p3_2.BondCon1);
  connect(J0p3_2.BondCon3,B6.fBondCon1);
  connect(B6.eBondCon1,C1.BondCon1);
  connect(J0p3_2.BondCon2,B7.eBondCon1);
  connect(B7.fBondCon1,R2.BondCon1);
end SimpleBondGraph;
```

Listing 3.2: Ideal Diode

```
model BondGraphs_RLC_circuit
  Modelica.BondLib.Electrical.Analog.Sources.SineVoltage sineVolt;
  Modelica.BondLib.Electrical.Analog.Basic.Resistor resistor;
  Modelica.BondLib.Electrical.Analog.Basic.Inductor inductor;
  Modelica.BondLib.Electrical.Analog.Basic.Capacitor capacitor;
  Modelica.BondLib.Electrical.Analog.Basic.Ground ground;
equation
  connect(capacitor.p, sineVolt.n);
  connect(sineVolt.p, resistor.p);
  connect(resistor.n, inductor.p);
  connect(inductor.n, capacitor.n);
  connect(ground.p, sineVolt.n);
end BondGraphs_RLC_circuit;
```

Listing 3.3: Ideal Diode

```
Model Diode   Ideal diode
    Extends TwoPin;
    Real s;
    Boolean off;
Equation
    Off  = s < 0;
    If off
        Then v=s;
    Else v=0;           // conditional equations
    End if;
     I = if off then 0 else s;  // conditional expression
End Diode;
```

Listing 3.4: Ideal Tunnel Diode

```
model IdealTunnelDiode "Ideal Tunnel Diode"
    extends Modelica.Electrical.Analog.Interfaces.OnePort;
    import Modelica.SIunits;
equation
    i = if v >= 0.0 and v <= 0.055
        then
        6.0105 * v^3 - 0.9917 * v^2 + 0.0545 * v
    elseif v >= 0.05 and v <= 0.35
        then
        0.0692 * v^3 - 0.0421 * v^2 + 0.004 * v + 0.000885794
    elseif v >= 0.35
        then
        0.2634 * v^3 - 0.2765 * v^2 + 0.0968 * v - 0.0112
    else 0;
end IdealTunnelDiode;
```

Listing 3.5: Ideal Tunnel Diode - Characterization circuit

```
model CharacteristicIdealTunnelDiodes
  Modelica.Electrical.Analog.Semiconductors.IdealTunnelDiode
    idealtunneldiode1;
  Modelica.Electrical.Analog.Basic.Resistor resistor1
    (R = 0.001);
  Modelica.Electrical.Analog.Basic.Ground
    ground1;
  Modelica.Electrical.Analog.Sources.SineVoltage
    sinevoltage1(V = 1.0, offset = 0);
equation
  connect(idealtunneldiode1.p,sinevoltage1.p);
  connect(sinevoltage1.n,ground1.p);
  connect(ground1.p,resistor1.n;
  connect(idealtunneldiode1.n,resistor1.p);
end CharacteristicIdealTunnelDiodes;
```

Listing 3.6: Tunnel Diode oscillator

```
model TunnelDiodeOscillator
  Modelica.Electrical.Analog.Basic.Inductor
    inductor1(L = 1e-06);
  Modelica.Electrical.Analog.Basic.Ground
    ground1;
  Modelica.Electrical.Analog.Sources.SineVoltage
    sinevoltage1(offset = 0.3);
  Modelica.Electrical.Analog.Basic.Resistor
    resistor1(R = 200);
  Modelica.Electrical.Analog.Semiconductors.IdealTunnelDiode
    idealtunneldiode1;
  Modelica.Electrical.Analog.Basic.Capacitor
    capacitor1(C = 1e-12);
equation
  connect(ground1.p,idealtunneldiode1.n);
  connect(capacitor1.n,idealtunneldiode1.n);
  connect(sinevoltage1.n,idealtunneldiode1.n);
  connect(inductor1.n,capacitor1.p);
  connect(inductor1.n,idealtunneldiode1.p);
  connect(resistor1.n,inductor1.p);
  connect(resistor1.p,sinevoltage1.p);
end TunnelDiodeOscillator;
```

Listing 3.7: Petri Net representation for Tunnel Diode

```
model TunnelDiodePetriNet
    PetriNets.Place11 Poff;     // Off state
    PetriNets.Place22 PcutOff;  // Cut-off state
    PetriNets.Place22 PnegRes;  // Negative Resistance state
    PetriNets.Place11 Psat;     // Saturation state

    PetriNets.Transition T1 (condLabel="V>0");
    PetriNets.Transition T2 (condLabel="V<0");
    PetriNets.Transition T3 (condLabel="V>Vpeak");
    PetriNets.Transition T4 (condLabel="V<Vpeak");
    PetriNets.Transition T5 (condLabel="V>Vvalley");
    PetriNets.Transition T6 (condLabel="V<Vvalley");
equation
    connect(Poff.outTransition, T1.inTransition);
    connect(Poff.inTransition, T2.outTransition);

    connect(PcutOff.outTransition1, T2.inTransition);
    connect(PcutOff.outTransition2, T3.inTransition);
    connect(PcutOff.inTransition1, T1.outTransition);
    connect(PcutOff.inTransition2, T4.outTransition);

    connect(PnegRes.outTransition1, T4.inTransition);
    connect(PnegRes.outTransition2, T5.inTransition);
    connect(PnegRes.inTransition1, T3.outTransition);
    connect(PnegRes.inTransition2, T6.outTransition);

    connect(Psat.outTransition, T6.inTransition);
    connect(Psat.inTransition, T5.outTransition);

end TunnelDiodePetriNet;
```

Chapter 4

Emulation of Analog Circuits

Analog emulation can also be comprehensively covered by using a model where the abstraction of the system is taken to a high level perspective. Examples of such methodology have been tested around the so called Programmable Analog Arrays, which use as a main block an arrangement of an operational amplifier with a set of configuration memories which store values related to external elements. Figure 4.1 presents a diagram with the main components of the PAA basic block.

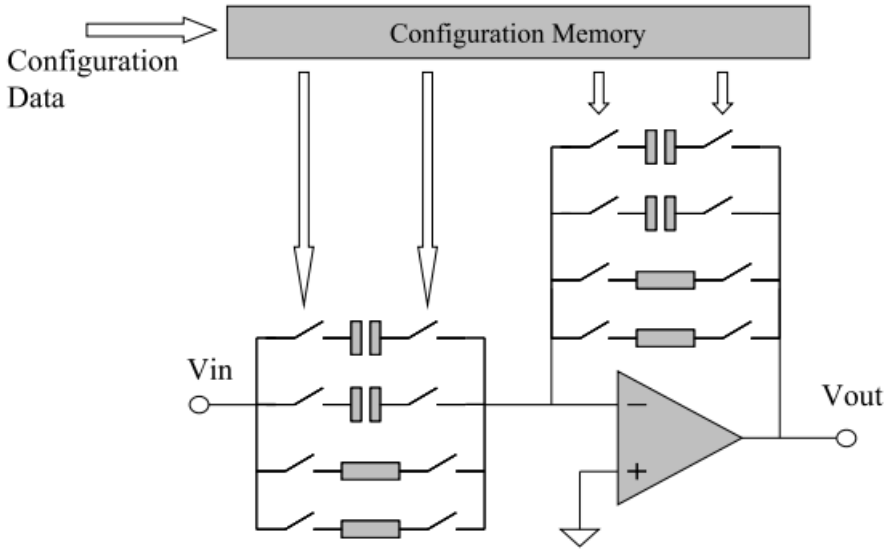

Figure 4.1: Programmable Analog Arrays - Basic Block [29]

4.1 Operational Amplifiers

4.1.1 General Description

A simple schematic model of an operational amplifier can be found in Figure 4.2. A simple model for an operational amplifier is described in Figure 4.3.

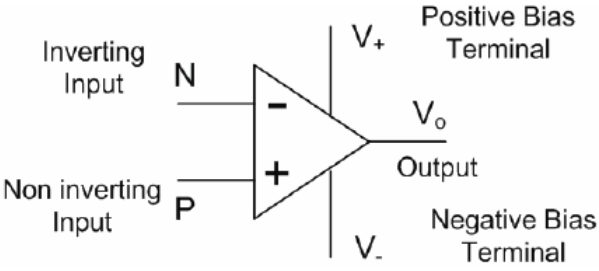

Figure 4.2: Operational Amplifier - Schematic representation [39]

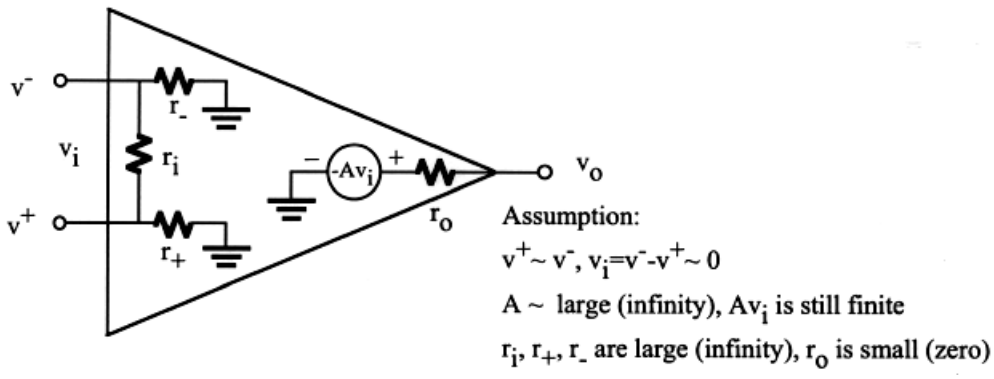

Figure 4.3: Operational Amplifier - Model [63]

4.1.2 Bond Graph Representation

Listing A.7 in Appendix A presents a general model for a basic operational amplifier. [10]. As shown in [39] and [23], operational amplifiers can

be described using Bond Graph models. Figure 4.4 shows a representation using a Bond Graph with Integral causality (BGI) approach to provide a more realistic model.

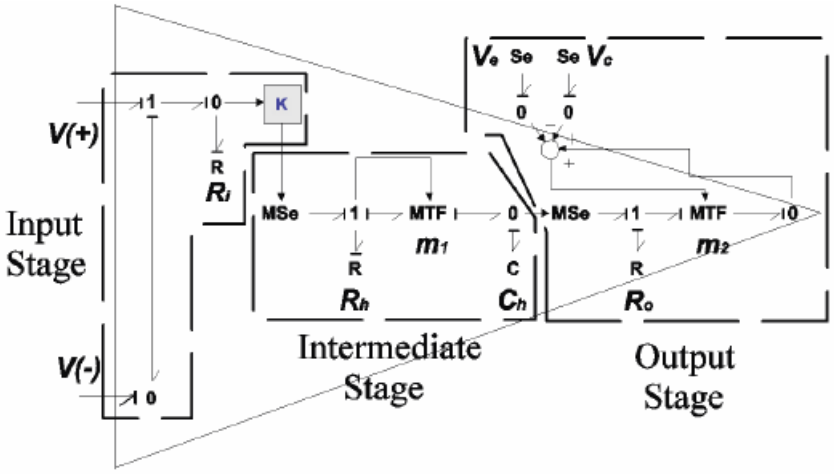

Figure 4.4: Operational Amplifier - Bond Graph representation [23]

4.2 Non-Inverting Amplifier

Figure 4.5 shows a simple functional description for a Non-Inverting amplifier. For the *positive* port of the operational amplifier, the input signal is used as the reference voltage, while the *negative* port is referenced to ground. R_f allows a direct control of the feedback voltage allowing the negative port to match the positive one.

R_f and R_{in} are set as a voltage divider and as a consequence the attained feedback voltage is represented as appears in Equation 4.1.

$$V_{feedback} = V_{out} \frac{R_{in}}{R_f + R_{in}} \tag{4.1}$$

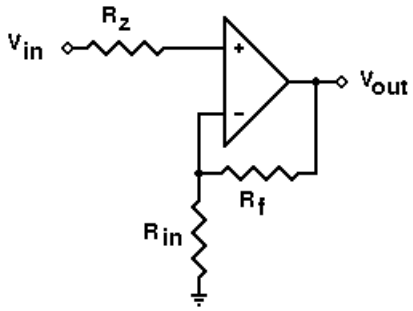

Figure 4.5: Operational Amplifier - Non-Inverting amplifier

4.2.1 Bond Graph Representation

Following the same assumption for BGI, it is possible to have a representation of a non-inverting amplifier as shown in Figure 4.6

4.3 Implementation Using FPAA

Using the theory related to Bond Graph & Petri Net representations, it is possible to generate an emulation model for verification of analog systems. The Configurable Analog Modules (CAM) can be arranged using the user interface. CAMs are predefined analog blocks present in the FPAA. AnadigmDesigner2 is the tool provided by the FPAA vendor to meet such need. Its simulator allows adding signal generators and oscilloscope probes.

Once the designer is satisfied with the results from the simulation, the device can be configured by downloading a configuration bit stream to the

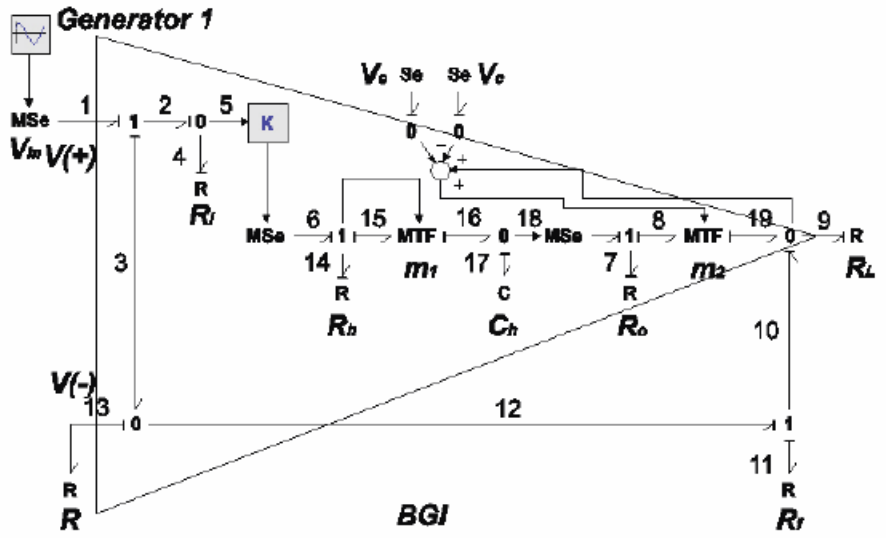

Figure 4.6: Non-Inverting Amplifier - Bond Graph representation

selected devices. AnadigmDesigner2 allows generating *C code* which can be used for dynamic configuration of the FPAA devices using external microprocessors. The parameters for the different CAM blocks can interactively be defined using a GUI interface.

Figure 4.8 shows a basic configuration for a Inverting Amplifier with a gain set to twice its input. Figure 4.7 shows the project settings. Figure 4.9 shows the results of the simulation after setting three probes in the circuit representation.

Figure 4.10 shows the CAM parameter interface for the *GainInv* block. The *C code* for such CAM module can be seen in Listing 4.1

Figure 4.11 shows the CAM parameter interface for the *InputCell* block. The *C code* for such CAM module can be seen in Listing 4.2

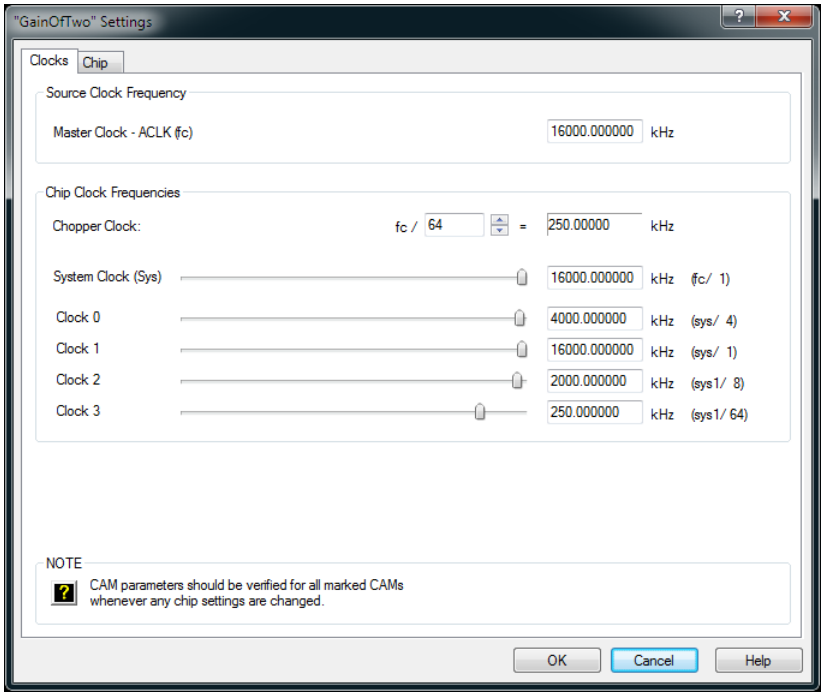

Figure 4.7: Implementation of a Inverting Amplifier using FPAAs - Project Settings

Figure 4.12 shows the CAM parameter interface for the *OutputCell* block. The *C code* for such CAM module can be seen in Listing 4.3

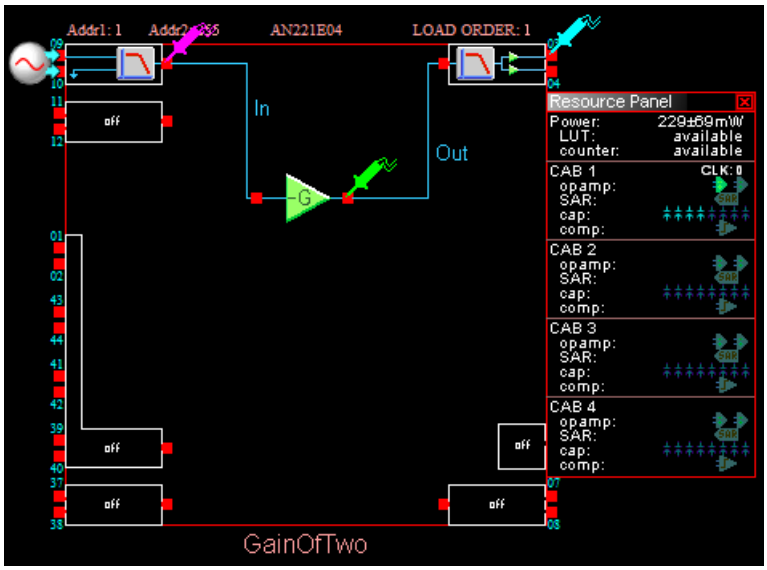

Figure 4.8: Implementation of a Inverting Amplifier using FPAAs

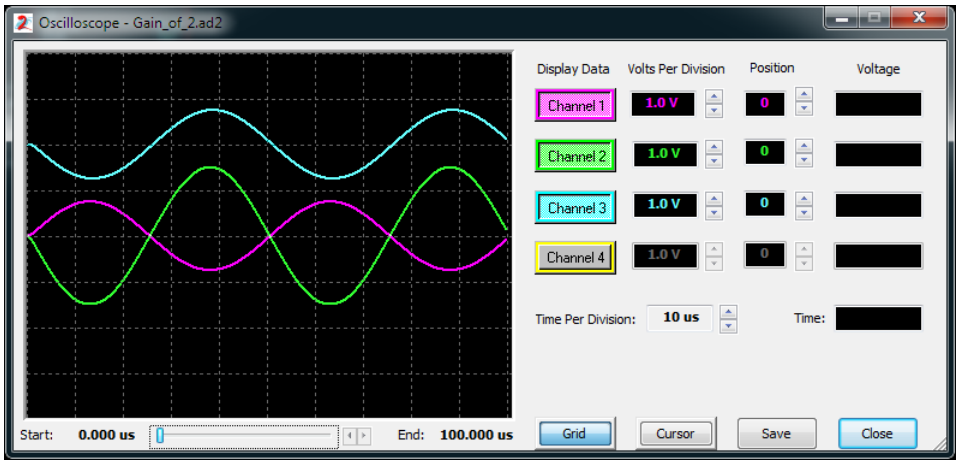

Figure 4.9: Implementation of a Inverting Amplifier using FPAAs - Simulation

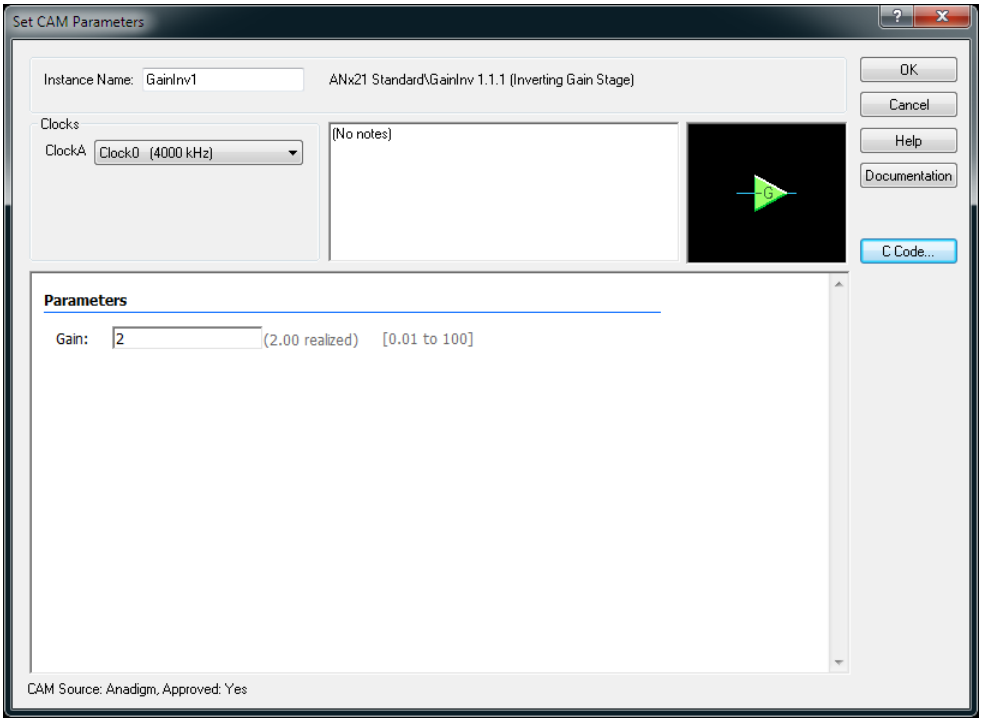

Figure 4.10: Implementation of a Inverting Amplifier using FPAAs - Gain Inv. block - CAM parameter configuration

Listing 4.1: Anadigm - Gain - CAM Parameters

```
/* A full fixed-point method for setting the gain of the module.
It accepts floating-point inputs and returns a floating-point
result. */
double fixed_setGAinInv( double G)

/* A full floating point method for setting the gain
of this module. */
double setGainInv( double G )
```

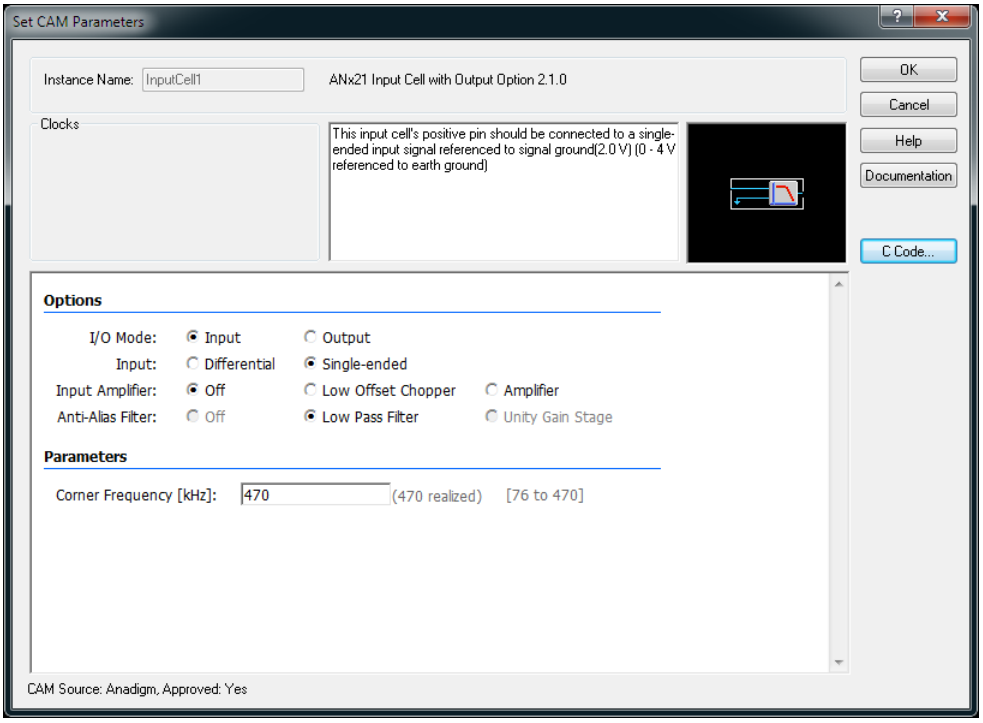

Figure 4.11: Implementation of a Inverting Amplifier using FPAAs -Input Cell block - CAM parameter configuration

Listing 4.2: Anadigm - Input Cell - CAM Parameters

```
/* This function sets the 3dB corner frequency of the filter
in this Input Cell. */
long setInputFilter( long F)
```

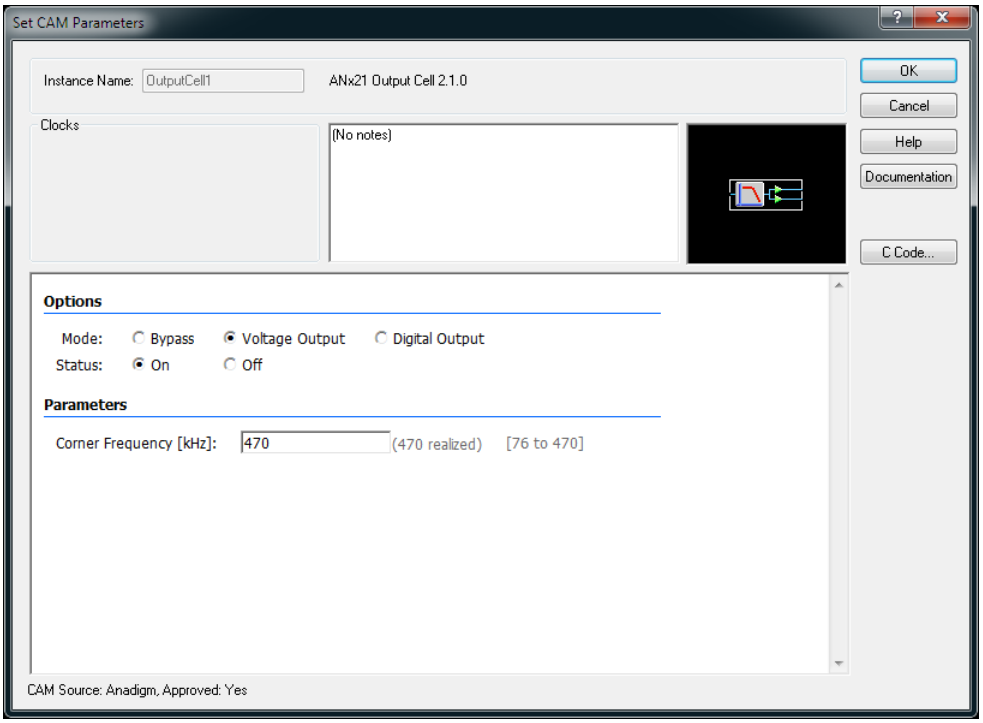

Figure 4.12: Implementation of a Inverting Amplifier using FPAAs - Output Cell block - CAM parameter configuration

Listing 4.3: Anadigm - Output Cell - CAM Parameters

```
/* This function sets the 3dB corner frequency of the filter
in this Output Cell. */
long setOutputFilter( long F)

/* This function can be used to turn a
"Voltage Mode Output Cell" ON (true) or OFF (0). */
void setOutputVoltageStatus( Bool status )
```

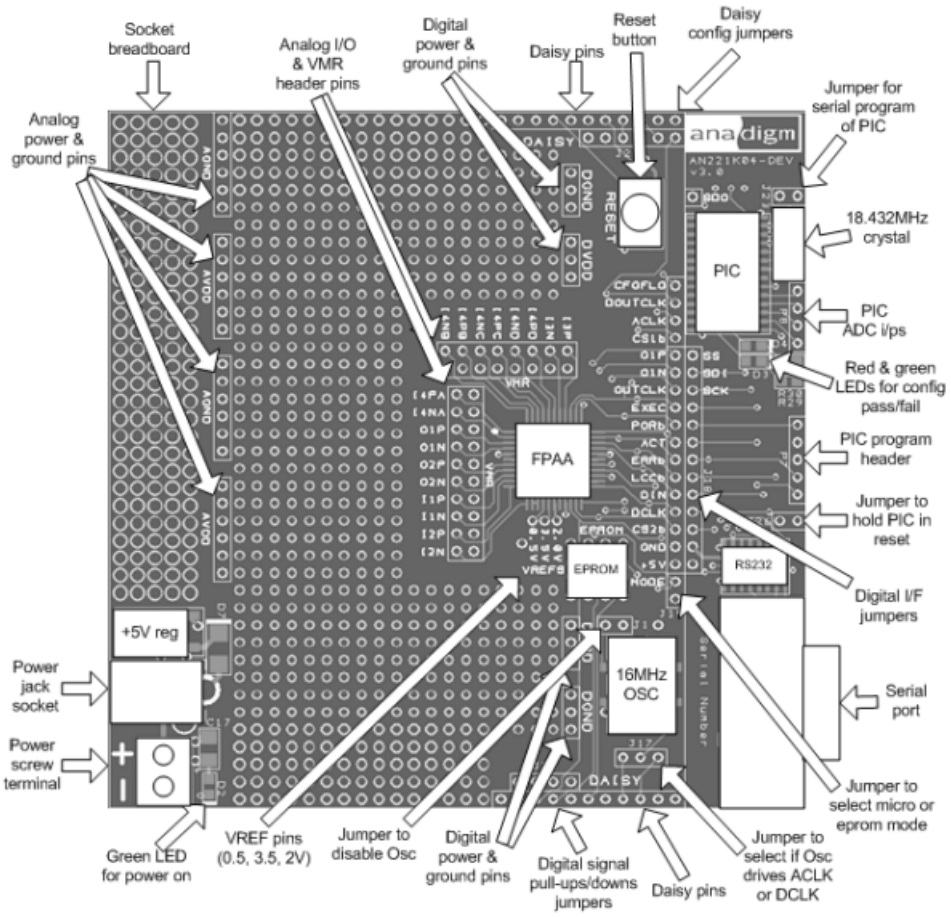

Figure 4.13: AnadigmVortex development board - Top-level layout[1]

Chapter 5

Conclusions

5.1 Summary of Key Contributions

Modeling alternatives have been presented as part of a wide range of mathematical descriptions. Among them *Stochastic Hybrid Systems*, *Petri Nets*, and *Bond Graph* models were described and analyzed.

Models were conceived as part of a overall and more complex system. Also, techniques for characteristics inheritance and object-oriented capabilities were described. A simple introduction to *Hierarchical Petri Nets* was done. Introduction to concepts related to Verification of analog systems was achieved by using practical examples.

Representation of functional models using high-level modeling environments such as *Open-Modelica* was carried out. All the Modelica descriptions can be simulated using the free Open-Modelica modeling suite. The necessary translations and modifications from libraries developed for commercial tools were done as part of the modeling setup.

Description of emulation methodologies using *Field Programmable Analog Arrays* was described using a practical example.

5.2 Future Work

5.2.1 Modeling

Modeling using *Bond Graph* and *Petri Nets* is still an open field for further exploration. There are many available tools capable of describing complex systems but the intricacy of the analog systems requires a deeper look into the details relevant to the stochastic nature of the systems.

Hierarchical modeling for multi-domain systems requires further exploration considering a non-deterministic individual behavior from the individual components.

5.2.2 Verification

Verification of analog systems can be achieved if object-oriented modeling tools are used as part of a modeling strategy where mathematical descriptions can be easily implemented. *Open-Modelica* offers a great set of resources to accomplish such task.

5.2.3 Emulation

The use of *FPAA* devices will have to be explored with more detail, and a methodical approach be implemented. The implementation of designs using graphical tools serves its purpose whenever the models do not require active updates. However, such an approach does not allow the integration of other tools foreign to the development suite. It would be really interesting to design a valid platform for a transparent translation between simple rules described

using say Bond Graph models and the required configuration files.

Appendices

Appendix A

Open Modelica

A.1 Basic Class

A Basic class can be described as shown in Listing A.1.[20]

Listing A.1: HelloWorld example

```
class HelloWorld "a simple equation"
    Real x(start=1);
equation
    der(x) = -x;
end HelloWorld;
```

The file has to be saved with a *.MO extension. To simulate the above class one has to use the Open Modelica Shell (OMShell) issuing the commands shown in Listing A.2

Listing A.2: HelloWorld - Commands for simulation

```
>> loadFile("provide path to the file")
>> simulate(HelloWorld, stopTime = 2)
>> plot(x)
```

After issuing the above commands the graph shown as Figure A.1

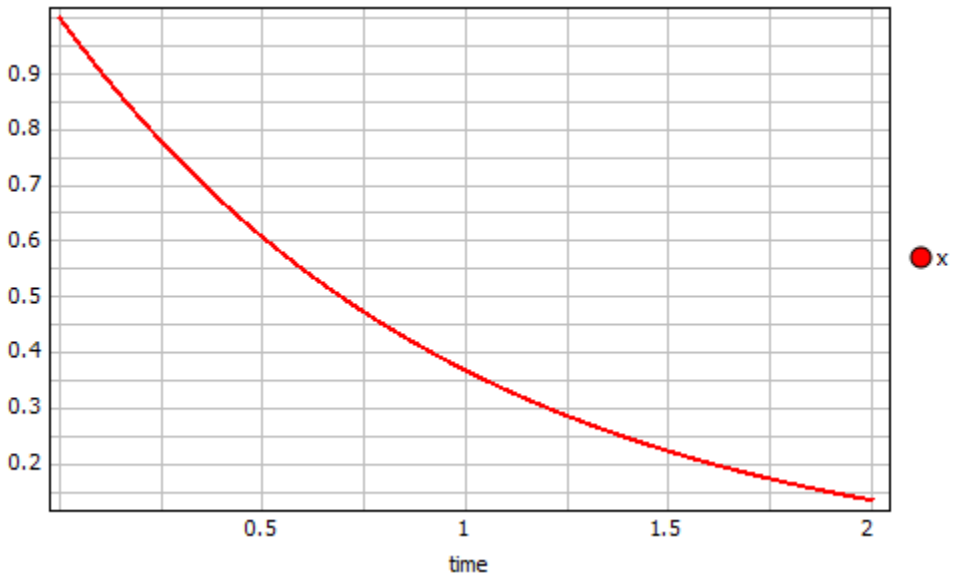

Figure A.1: Basic example's plot

A.2 Algebraic Equations

Model for algebraic equations with no derivatives is shown in Listing A.3

Listing A.3: Algebraic Equations

```
class DAEexample
    Real x(start=0.9);
    Real y;
equation
    der(y) + (1 + 0.5 * sin(y)) * der(x) = sin(time);
    x - y = exp(-0.9 * x) * cos(y);
end DAEexample;
```

The resulting graph can be found as Figure A.2

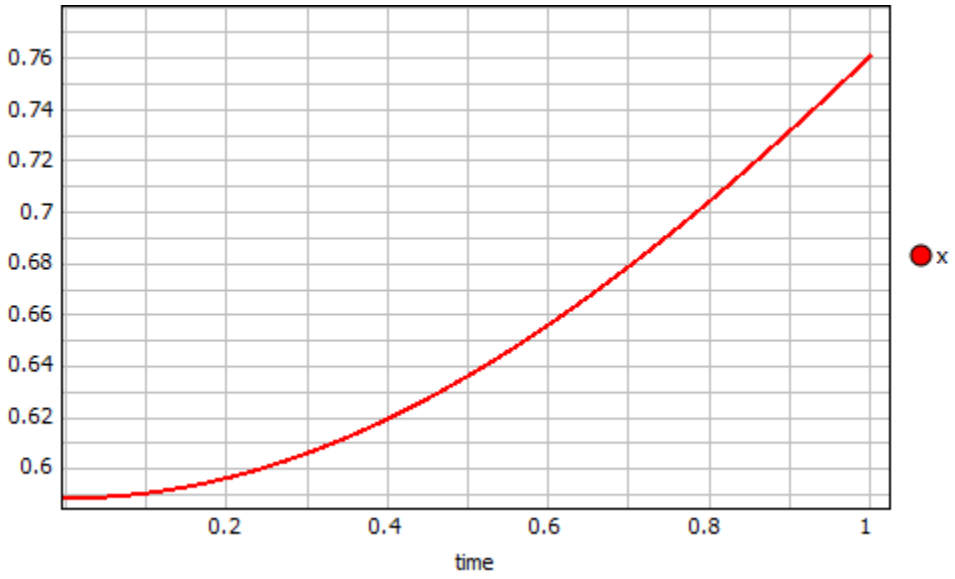

Figure A.2: Algebraic Equation's plot

A.3 Van der Pol Oscillator Model

Listing A.4 shows a simple model for Van der Pol oscillator

The resulting graph can be found as Figure A.3

To simulate issue the commands shown in Listing A.5

Listing A.5: Van der Pol - Commands for simulation

```
>> loadFile("provide path to the file")
>> simulate(VanDerPol, stopTime = 25)
>> plotParametric(x,y)
```

Listing A.4: Van der Pol oscillator

```
class VanDerPol  "Van der Pol oscillator model"
    // 'x' starts at 1
    Real x(start = 1)  "Descriptive string for x";
    // 'y' starts at 1
    Real y(start = 1)  "y coordinate";
    parameter Real lambda = 0.3;
equation
    // 1st diff equation
    der(x) = y;
    // 2nd diff equation
    der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

A.4 Ideal Tunnel Diode - Complete Model

Complete model for the ideal tunnel diode including version, documentation and icon can be found in Listing A.6

Listing A.6: Ideal Tunnel Diode - Complete Model

```
model IdealTunnelDiode "Ideal Tunnel Diode"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  import Modelica.SIunits;
  annotation(Diagram(graphics = {Polygon(points = {{30,0},
  {-30,40},{-30,-40},{30,0}}, rotation = 0,
   lineColor = {0,0,0}, fillColor = {255,0,0},
    pattern = LinePattern.Solid, fillPattern = FillPattern.None,
     lineThickness = 0.25),Line(points = {{-96,0},{40,0}},
      rotation = 0, color = {0,0,255},
      pattern = LinePattern.Solid,
      thickness = 0.25),Line(points = {{40,0},{96,0}},
      rotation = 0, color = {0,0,255},
      pattern = LinePattern.Solid,
      thickness = 0.25),Line(points = {{30,40},{30,-40}},
      rotation = 0, color = {0,0,255},
      pattern = LinePattern.Solid, thickness = 0.25)}));

```

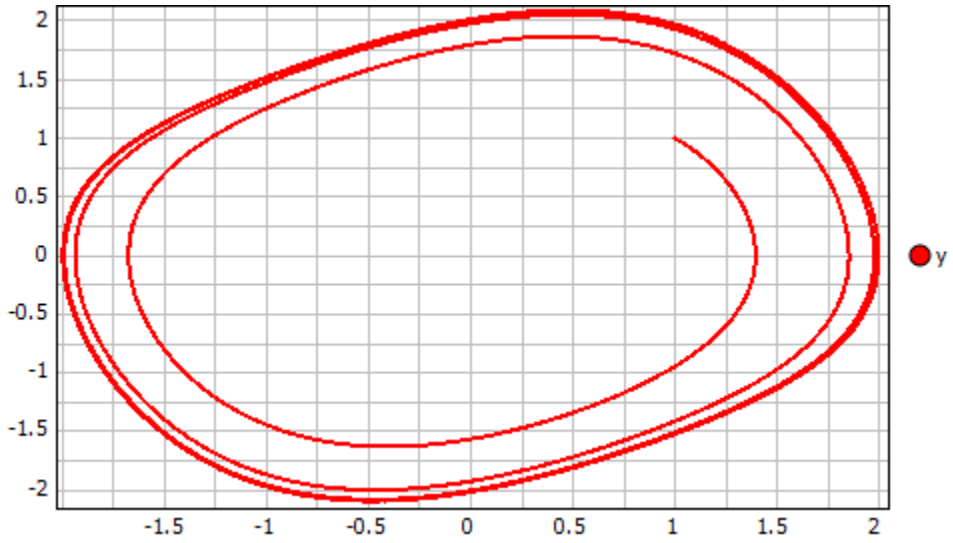

Figure A.3: Van der Pol oscillator model

```
equation
  i = if v >= 0.0 and v <= 0.055 then
   6.0105 * v ^ 3 - 0.9917 * v ^ 2 + 0.0545 * v
    elseif
    v >= 0.05 and v <= 0.35
    then
    0.0692 * v ^ 3 - 0.0421 * v ^ 2 + 0.004 * v + 0.000885794
    elseif
    v >= 0.35 then 0.2634 * v ^ 3
            - 0.2765 * v ^ 2 + 0.0968 * v - 0.0112
      else 0;
  annotation(Documentation(info = "
<HTML>
<P>
The ideal tunnel diode model is described as a one port entity.
The diode formula is:
</P>
<PRE>

  i =   if (0.000 <= Vtd <= 0.055) then
    6.0105*v^3 - 0.9917*v^2 + 0.0545*v
         elsif (0.055 <= Vtd <= 0.350) then
    0.0692*v^3 - 0.0421*v^2 + 0.0040*v + 0.00088579
         elsif (0.350 <= Vtd) then
```

```
      0.2634*v^3 - 0.2765*v^2 + 0.0968*v - 0.0112
        else
            0;
</PRE>
<P>
The formula has been taken from Mark Greenstreet's paper
[M.R. Greenstreet. Verifying VLSI Circuits.Dep. of CS.
Univ. of BC, Canada]

</P>
</HTML>
", revisions = "<html>
<ul>
<li><i> November 11, 2011   </i>
       by Ricardo Ramirez<br> implemented<br>
       </li>
</ul>
</html>"), Icon(coordinateSystem(preserveAspectRatio
= true, extent = {{-100,-100},{100,100}},
grid = {1,1}),
graphics = {Polygon(points = {{30,0},{-30,40},
{-30,-40},{30,0}}, lineColor = {0,0,0},
 fillColor = {255,255,255},
 fillPattern = FillPattern.Solid),
 Line(points = {{-90,0},{40,0}}, color = {0,0,255}),
 Line(points = {{40,0},{90,0}}, color = {0,0,255}),
 Line(points = {{30,40},{30,-40}}, color = {0,0,255}),
 Text(extent = {{-150,-49},{149,-77}},
  lineColor = {0,0,0}, textString = "Vt=%Vt"),
  Text(extent = {{-154,100},{146,60}},
  textString = "%name", lineColor = {0,0,255}),
  Line(visible = useHeatPort, points = {{0,-100},{0,-20}},
   color = {127,0,0}, smooth = Smooth.None,
   pattern = LinePattern.Dot)}),
   Diagram(coordinateSystem
   (preserveAspectRatio = true,
   extent = {{-100,-100},{100,100}}, grid = {1,1}),
    graphics = {Polygon(points = {{30,0},{-30,40},
    {-30,-40},{30,0}}, lineColor = {0,0,0},
    fillColor = {255,0,0}, fillPattern = FillPattern.None),
    Line(points = {{-96,0},{40,0}}, color = {0,0,255}),
    Line(points = {{40,0},{96,0}}, color = {0,0,255}),
    Line(points = {{30,40},{30,-40}}, color = {0,0,255})})));
end IdealTunnelDiode;
```

A.5 Operational Amplifier

A model for an operational amplifier can be found in Listing A.7

Listing A.7: Modelica - Operational Amplifier

```
model OpAmp "Operational amplifier"
  parameter Boolean enforceStates = true
  "Use electrical variables as states";

  Modelica.Electrical.Analog.Interfaces.Pin vi_n;
  Modelica.Electrical.Analog.Interfaces.Pin vi_p;
  Modelica.Electrical.Analog.Interfaces.Pin vo;
  Modelica.Electrical.Analog.Interfaces.Pin vcc;
  Modelica.Electrical.Analog.Interfaces.Pin vee;
  Electrical.Analog.Semiconductors.PNP
    PNP1(enforceStates=enforceStates);
  Electrical.Analog.Basic.Resistor R1(R=1.5e6);
  Electrical.Analog.Semiconductors.PNP
    PNP2(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.PNP
    PNP3(enforceStates=enforceStates);
  Electrical.Analog.Basic.Capacitor C1(C=10e-12);
  Electrical.Analog.Semiconductors.NPN
    NPN4(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.NPN
     NPN5(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.PNP
    PNP6(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.PNP
    PNP7(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.NPN
    NPN8(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.PNP
    PNP9(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.NPN
    NPN10(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.NPN
    NPN11(enforceStates=enforceStates);
  Electrical.Analog.Semiconductors.NPN
    NPN12(enforceStates=enforceStates);

equation
```

```
  connect(PNP1.C, R1.p);
  connect(PNP6.B, vi_n);
  connect(PNP6.E, PNP7.E);
  connect(PNP2.C, PNP7.E);
  connect(PNP7.B, vi_p);
  connect(C1.p, NPN10.B);
  connect(PNP6.C, NPN11.C);
  connect(NPN11.C, NPN11.B);
  connect(R1.n, NPN11.E);
  connect(NPN12.B, NPN11.B);
  connect(NPN10.E, NPN12.E);
  connect(NPN11.E, NPN12.E);
  connect(NPN12.E, vee);
  connect(PNP7.C, NPN12.C);
  connect(NPN12.C, NPN10.B);
  connect(PNP9.C, NPN10.E);
  connect(PNP1.E, PNP2.E);
  connect(PNP2.E, vcc);
  connect(NPN4.C, PNP3.E);
  connect(PNP2.E, PNP3.E);
  connect(PNP3.C, NPN5.C);
  connect(NPN5.C, NPN4.B);
  connect(NPN5.B, NPN5.C);
  connect(NPN5.E, NPN8.C);
  connect(NPN8.B, NPN8.C);
  connect(NPN4.E, PNP9.E);
  connect(PNP9.E, vo);
  connect(NPN8.E, NPN10.C);
  connect(C1.n, NPN10.C);
  connect(PNP9.B, NPN10.C);
  connect(PNP1.B, PNP2.B);
  connect(PNP1.B, PNP1.C);
  connect(PNP2.B, PNP3.B);
end OpAmp;
```

Appendix B

VLSI Models

B.1 VHDL-AMS Models

B.1.1 Buck Converter Without Feedback

Listing B.1 shows a representation of a Buck Converter using VHDL-AMS. Notice it is a simulation model and it is not synthesizable.

Listing B.1: Buck Converter - VHDL-AMS model

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  ------------------------------------------------
5  entity tb_BuckConverter is
6      port (ctrl : std_logic );
7  end tb_BuckConverter;
8  ------------------------------------------------
9  architecture tb_BuckConverter of tb_BuckConverter is
10     terminal vin : electrical;
11     terminal vmid : electrical;
12     terminal vout : electrical;
13 begin
14     L1: entity work.inductor(ideal)
15         generic map ( ind => 6.5e-3 )
16         port map ( p1 => vmid, p2 => vout );
17     -------------------
18     C1: entity work.capacitor(ideal)
19         generic map ( cap => 1.5e-6 )
20         port map ( pos => vout, p2 => electrical_ref );
21     -------------------
22     VinDC:entity work.v_constant( ideal )
```

```
        generic map ( level => 42.0 )
        port map ( pos => vin, neg => electrical_ref );
    -------------------
     RLoad: entity work.resistor( ideal )
        generic map ( res => 2.4 )
        port map ( p1 => vout, p2 => electrical_ref );
    -------------------
    D1: entity work.diode( ideal )
        port map ( p => electrical_ref, n => vmid );
    -------------------
    sw1:    entity work.switch_dig( ideal )
        port map ( sw_state => ctrl, p2 => vmid, p1 => vin );
end architecture tb_BuckConverter;
```

B.1.2 Diode

An example for a description of a simple Ideal Diode can be found at Listing B.2. [49]

Listing B.2: Diode - VHDL-AMS model

```

library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
-- FUNDAMENTAL_CONSTANTS package needed for
-- Boltzmann constant
-- (PHYS_K = Joules/Kelvin) and electron charge
-- (PHYS_Q = coulomb)
use IEEE.FUNDAMENTAL_CONSTANTS.all;

entity diode is
    generic (Isat: current := 1.0e-14); -- Sat. current [Amps]
    port (terminal p, n : electrical);
end entity diode;

architecture ideal of diode is
    -- Declare internal quanties and constants
    quantity v across i through p to n;
    constant TempC : real := 27.0; -- Ambient Temp. [Degrees]
```

```
    constant TempK : real := 273.0 + TempC; -- Temp. [Kelvin]
    constant vt : real := PHYS_K*TempK/PHYS_Q; -- Thermal Volt.

-- This function is to limit the exponential function to avoid
-- convergence problems due to numerical overflow.
-- At x=100, it becomes a straight line
-- with slope matching that at the intercept.
    function limit_exp( x : real ) return real is
        variable abs_x : real := abs(x);
        variable result : real;
    begin
        if abs_x < 100.0 then
            result := exp(abs_x);
        else
            result := exp(100.0) * (abs_x - 99.0);
        end if;
    -- If exponent is negative, set exp(-x) = 1/exp(x)
    if x < 0.0 then
        result := 1.0 / result;
    end if;
    return result;
    end function limit_exp;
begin -- ideal architecture
    -- Fundamental equation
    i == Isat*(limit_exp(v/vt) - 1.0);
end architecture ideal;
```

Appendix C

Tunnel Diode

The tunnel diode (some times called Esaki diode after its inventor) is a P-N junction diode heavily doped (1000 times greater than conventional diodes). [3]

C.1 Characteristics

The device is heavily doped which makes the width of the depletion layer to be proportionally much more smaller than the one present in a convention diode in a 1 to 10,000 ratio approx. As a consequence, the breakdown voltage is reduced to a very small value.

The behavior of the careers is altered allowing *punch through* without the requirement of energy through the potential barrier. As a consequence, the Esaki diode allows forward current even for small voltages (which are usually verified in the range of $100mV$). This quantum phenomenon is called resonant tunneling.

The tunnel diode has some special features which are useful in specific applications. It has a triggering voltage V_{peak}. It also has a very low triggering current, which provides a high pulse current capability. Its unique negative

resistance quantum characteristic provides a special framework for low power applications.

C.2 Fabrication

Esaki diodes are usually fabricated using either Germanium (Ge), Gallium arsenide (GaAs) or Gallium antimodine (GaSb). Figure C.1 shows some of the Doping profiles used in its fabrication. Silicon is not used since the ration between *peak current* and *valley current* is very small.

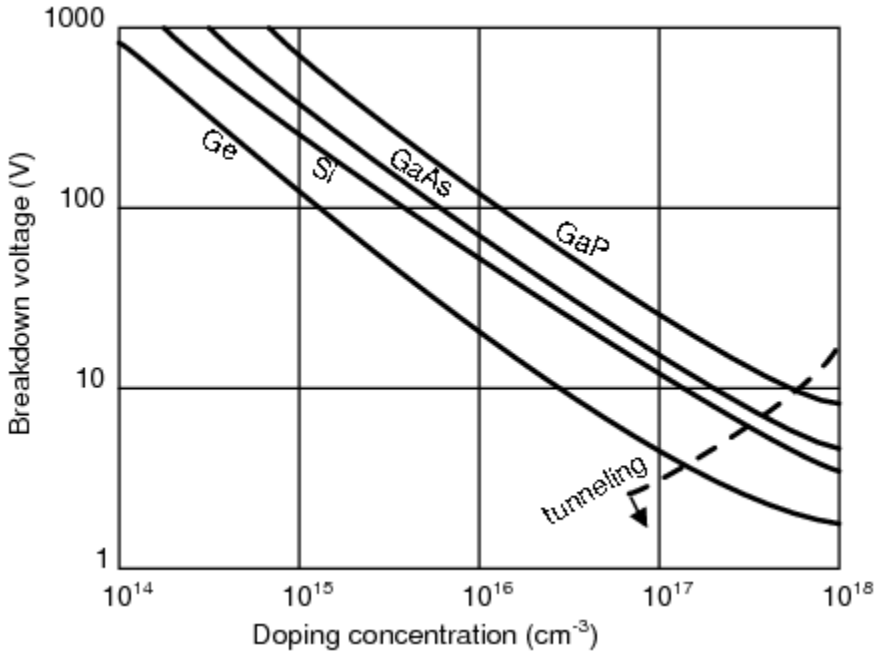

Figure C.1: Tunnel diode - Doping profiles

C.3 Issues

Since it is a *low power* device it can easily be damaged by either Electrostatic discharges (ESD) or heat.

Bibliography

[1] AN221K04-v3: AnadigmVortex Development board. Technical report, Anadigm, 2006.

[2] *Lecture Notes in Artificial Intelligence*, volume 4334, chapter Verification of Object-Oriented Software: the KeY approach. Springer, 2007.

[3] Tunnel diode. http://www.circuitstoday.com/tunnel-diode, August 2009.

[4] B. Aghdaie and B. Sheu. Overcoming limitations of lumped mos models. *Circuits and Devices magazine, IEEE*, 16(2):19–26, 2000.

[5] M.E. Brinson, S. Jahn, and H. Nabijou. QUCS, SPICE and Modelica equation-defined modeling techniques for the construction of compact device models based on a common model template structure. In *Frontiers of Compact Modeling for advanced Analog/RF applications*, April 2011.

[6] Jan F. Broenink. Introduction to Physical Systems modeling with Bond Graphs. Technical report, University of Twente, Control Laboratory, 1999.

[7] Manuela Bujorianu. Stochastic hybrid systems overview. http://personalpages.manchester.ac.uk/staff/Manuela.Bujorianu/SHS_survey04.pdf, December 2008.

[8] R. Caverly. *CMOS RFIC: Design Principles*. Artech House, 2007.

[9] Francois E. Cellier. Mathematical modeling of physical systems: Electrical circuits I, September 2011.

[10] Francois E. Cellier and Angela Nebot. The Modelica Bond Graph library. Proc. 4th Modelica Conference, Hamburg, Germany, 2005.

[11] P. Ciampolin, L. Roselli, and G. Sopponi. Mixed-mode circuit simulation with full-wave analysis of interconnections. *Electron Devices, IEEE Transactions on*, 44(11):2098–2105, nov 1997.

[12] David Malo Cid. OpenModelica for Analog IC Design. Master's thesis, Linkoping University, August 2011.

[13] C. Claus, A. Schneider, T. Leitner, and P. Schwarz. Modeling of electrical circuits with modelica. Workshop Modelica, Lund, Sweden, 2000.

[14] Universitat des Saarlandes. Stochastic hybrid systems. http://depend.cs.uni-sb.de/index.php?444, winter 2006.

[15] M. Edwards and E. Wasserman. Transient simulations of rf and high data rate circuits. Technical report, ARMMS: RF and Microwave society, 2003.

[16] M. J. Bertin et al. *Pisot and Salem Numbers*. user Verlag, Berlin, 1992.

[17] Stefan Fabricius. Extensions to the Petri Net library in Modelica. https://www.modelica.org/libraries/ExtendedPetriNets, December 2001.

[18] Paul Fishwick. Computer simulation syllabus: Petri nets. http://www.cise.ufl.edu/~fishwick/cap4800/, 2001.

[19] Peter Fritzson. *Principles of Object-Oriented modeling and simulation with Modelica 2.1.* IEEE press, 2004.

[20] Peter Fritzson. Introduction to object-oriented modeling and simulation with open modelica: Tutorial. Technical report, 2006.

[21] Rob J. van Glabbeek. Bounded non-determinism and the approximation induction principle in process algebra. http://theory.stanford.edu/people/rvg/abstracts.html, 1987.

[22] Carla Gomes. Info 2950: Mathematical Methods for Information Science. http://www.infosci.cornell.edu/courses/info2950/2011sp/, 2011.

[23] Gilberto Gonzalez and Roberto Tapia. Operational Amplifiers and Active Filters: A Bond Graph Approach. http://www.intechopen.com/articles/show/title/operational_amplifiers_and_active_filters__a_bond_graph_approach, May 2008.

[24] M Goosens, F Mittelbach, and A Samarin. *The LaTeXCompanion.* Addison-Wesley, 1994.

[25] Mark R. Greenstreet. Verifying vlsi circuits. *Lecture Notes in Computer Science: Automated technology for Verification and Analysis*, 5799:1–20, 2009.

[26] Jan Friso Groote. Process Algebra and Structured Operational semantics. Master's thesis, University of Amsterdam, 1991.

[27] Thomas A. Henzinger. The Theory of Hybrid Automata. IEEE Symposium on Logic in Computer Science, pages 278–292, 1996.

[28] J.P. Hespnha. Hybrid and switched systems: Simulation of hybrid systems. http://www.ece.ucsb.edu/~hespanha/ece229, 2005.

[29] J. Kampe, M. Ponca, U. Heiber, A. Rummler, and C. Wisser. Fast mixed-signal system prototyping using unique programmable analog array. 2005.

[30] J. Kampe, G. Scarbata, S. Arlt, U. Heiber, M. Ponca, A. Rummler, and C. Wisser. Rapid development kit for mixed signal systems (rdk). Sofia, Bulgaria, 2004. 27th Int'l Spring Seminar on Electronics Technology (ISSE).

[31] Donald K. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[32] Philip T. Krein. *Elements of Power Electronics*. Oxford University Press, 1998.

[33] N.M. Kriplani, S. Bowyer, J. Huckaby, and M.B. Steer. Modeling of an esaki tunnel diode in a circuit simulator. *Active and Passive Electronic Components*, 2011(Article ID 830182), 2011.

[34] K. Kuwabara, N. Sato, T. Shimamura, H. Morimura, J. Kodate, and H. Ishii. Integrated rf-mems technology for reconfigurable transceivers. *NTT Technical review*, 2008.

[35] Leslie Lamport. *LaTeX: A document preparation system.* Addison-Wesley, 2nd edition, 1994.

[36] Thomas H. Lee. *The Design of CMOS Radio-Frequency Integrated Circuits*, chapter Distributed Systems, pages 202–220. Cambridge University Press, 2004.

[37] S. Little, D. Walter, N. Seegmiller, C. Myers, and T. Yoneda. Verification of Analog and Mixed-Signal circuits using Timed Hybrid Petri Nets. *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2004.

[38] Scott R. Little. *Efficient Modeling and Verification of Analog and Mixed-Signal circuits using Labeled Hybrid Petri Nets.* PhD thesis, The University of Utah, December 2008.

[39] C.Peraza M., J.G. Diaz, F. Arteaga, C. Villanueva, and F. Gonzalez-Longatt. Modeling of faults in operational amplifier circuits using Bond Graph. IECON 2008. 34th Annual Conference of IEEE Industrial Electronics, 2008., 2008.

[40] Ka L. Man and Tomas Krilavicius. *Intelligent Automation and Computer Engineering*, chapter Research on process algebraic analysis tools

for Electronic Systems Design, pages 415–427. Springer, 2010.

[41] Ka L. Man and M. P. Schellekens. Analysis of a mixed-signal circuit in hybrid process algebra acp. *Engineering Letters*, 15(2):1–20, 2007.

[42] Gurmeet Singh Manku, Ramin Hojati, and Robert Brayton. Structural Symmetry and Model Checking. volume 1427 of *Lecture Notes in Computer Science*, pages 159–171, 1998.

[43] F. Martin, X. Oriols, and J. Garcia-Garcia. Comparison of distributed and lumped element models for analysis of filtering properties of nonlineartransmission lines. *International journal of RF and microwave computer-aided engineering*, 12(6):503–507, 2002.

[44] Ian M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. Technical report, 2007.

[45] P.J. Mosterman, M. Otter, and H.Elmqvist. Modeling Petri Nets as local Constraint equations for Hybrid Systems using Modelica. *Modelica*, 1998.

[46] Sachin J. Mujumdar. CS 367 Theory of Hybrid Automata. Technical report, 2002.

[47] US Navy. *Navy Electricity and Electronics Training Series (NEETS)*, chapter Chapter 3 - Module 7 - Introduction to solid-state devices and power supplies, pages 51–55.

[48] J. Oehlerking and O. E. Theel. A decompositional proof scheme for automated convergence proofs of stochastic hybrid systems. In *Automated Technology for Verification and Analysis (ATVA)*, 2009.

[49] School of Electronics and University of Southampton Computer Science. Switch-mode power regulator. http://www.syssim.ecs.soton.ac.uk/vhdl-ams/examples/smpr.htm, August 2005.

[50] Andre Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.

[51] Andre Platzer. Logic and compositional verification of hybrid systems. Technical report, 2011.

[52] Alf J. van der Poorten. Some problems of recurrent interest. Technical Report 81-0037, School of Mathematics and Physics, Macquarie University, North Ryde, Australia 2113, August 1981.

[53] Fernando Rangel. Radio frequency electronics: Transmission lines and two-port networks. http://personalpages.manchester.ac.uk/staff/Manuela.Bujorianu/SHS_survey04.pdf, December 2008.

[54] Hector J. De Los Santos. *RF MEMS circuit design: for wireless communications*. Artech House, 2002.

[55] Robert M. Shapiro. Validation of a vlsi chip using hierarchical colored Petri Nets. *Microelectronics Reliability*, 31(4):607–625, 1991.

[56] R.K. Shyamasundar. Compositional approach for system design: Semantics of systemc. Technical report, IBM Research, India Research lab, and Tata Institue of fundamental research, 2005.

[57] N. Sidorova, M. Voorhoeve, and J. v.d. Woude. A calculus of Petri Net components: Semantics for Petri nets with hierarchy. Technical report, Aarhus University, Denmark, 2001.

[58] Grigory Simin. ELCT 563: Semiconductor devices lectures. http://www.ee.sc.edu/personal/faculty/simin, 2009.

[59] A. Sinha, A. Mondal, and K. Modi. Introduction to hybrid automata. Technical report, IIT KGP: Formal-V Group, 2005.

[60] Michael Spivak. *The joy of TEX*. American Mathematical Society, Providence, R.I., 2nd edition, 1990.

[61] Wenguan Sui. *Time-Domain Computer Analysis of Nonlinear Hybrid Systems*. CRC Press, 2001.

[62] Concordia University. Hardware Verification Group: Analog and Mixed Signal Analysis. http://hvg.ece.concordia.ca/projects/ams, 2012.

[63] Ruye Wang. Operational amplifier circuits. http://fourier.eng.hmc.edu/e80/active_filter/node7.html, 2008.

[64] Louis Woods. Data Processing on Modern Hardware: Regular Expressions in Hardware. http://www.systems.ethz.ch/education/past-courses/hs09/mmdbms, 2009.

Vita

Ricardo Ramírez-Gómez received the Bachelor of Science degree in Electrical Engineering from the Universidad Nacional de Colombia in 2000. The same year, he was selected to join the Fermi National Accelerator Lab's scientific collaboration (Batavia, IL USA). There, he worked for more than four years designing and testing instrumentation for high-energy physics experiments at two of Fermilab's Project D-Zero's particle detectors: the Central Fiber Tracker and the Forward Proton detector. In 2005, he received a Masters of Science degree in Electrical Engineering with emphasis in embedded systems from the University of Texas at Arlington. He has worked for a handful of companies developing embedded applications for NASA, USA Department of Defense (MDA, Army's NSSC, Navy's ONR, and Air Force's AFRL), and USA Dep. of Energy. He is a licensed engineer registered with the Texas Board of Professional Engineers and an amateur radio enthusiast.

Permanent address: PO Box 8525
Austin, Texas 78713

This report was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.